卒業論文

my_help の class 選択の Strategy 化

関西学院大学理工学部

情報科学科 西谷研究室

27017557 星野 壮哉

2023 年 3 月

概 要

my_help はエディターとして CUI(Character User Interface) である Emacs での利用を想定して作られていた. そのため，テキストの保存フォーマットとしてはマークアップ言語の org-mode(org) を使っていた. しかし, 多くのユーザーに使われるためには, GUI(Graphical User Interface) での操作が可能な VSCode とそのデフォルトマークアップ言語である Markdown(md) の組み合わせに変更する必要がある．このために佐々木は保存フォーマットを Markdown とする機能を my_help に追加している [1]. 当初, List 表示を実行するクラスにおいてそのマークアップ言語を実装されていた. これを, マークアップ言語から一度 Hash に読み込む改良を加えた. これによって, それぞれのマークアップ言語から Hash に読み込む Org2Hash と Md2Hash の 2 つのクラスが存在し, それを if 文で選択していた. しかし, if 文による条件分岐の選択は複雑なコードになる傾向がある. そのため, State あるいは Strategy パターンに変更した. こうすることによってメンテナンスしやすいコードに変更することができた.

目 次

図 目 次

第1章　序論

my_help とは, CUI ヘルプの MAN(マニュアル) の NAME(名前), SNOPSIS(概要), DESCRIPTON(説明), ... といった形の Usage(用法) 出力を真似て, user 独自の help を作成・提供することで, 用語の説明を書き取りやすく, また見直しやすくするターミナル上での使用を前提としたアプリである [2]. Ruby のアプリ, ライブラリ管理パッケージの gem で公開されている. gem とは RubyGems が公開している Ruby のパッケージである.[3].

現在までのところ, my_help はエディターとして CUI である Emacs での利用を想定して作られていた. そのため, テキストの保存フォーマットとしてはマークアップ言語の org-mode を使っていた. しかし, 初心者などのより多くのユーザーに使われるためには, GUI での操作が可能な VSCode とそのデフォルトマークアップ言語である Markdown の組み合わせに変更する必要がある.

このために佐々木は保存フォーマットを Markdown とする機能を my_help に追加している [1]. 当初, List 表示を実行するクラスにおいてそのマークアップ言語の選択が実装されていた. これを, マークアップ言語から一度 Hash に読み込む改良を加えた. これによって, それぞれのマークアップ言語から Hash に読み込む Org2Hash と Md2Hash の 2 つのクラスが存在した. それぞれのクラスをパラメータに従って選択する実装がなされていた.

しかし, if 文による条件分岐の選択は複雑なコードになる傾向がある. このような場合には, State あるいは Strategy パターンに変更する事が推奨されている [4, p.261]. そこで本研究ではこの条件分岐選択を State あるいはそれと同等の構造に変更することを目的とする.

第2章　手法

2.1　my_help

my_help は独自の help を提供することを目的としている．西谷研究室では講義の内容や一度参考にしたサイトのリンクなどをメモすることで活用している．

my_help で提供している表示機能には，item(その言葉の要素) の表示，簡単な解説，省略オプションなどがある．

2.1.1　my_help の使い方

my_help のコマンドの一覧は以下のコマンドをターミナル上で入力すると出力される．

```
> my_help
Commands:
  my_help delete [HELP]       # delete HELP
  my_help edit [HELP]         # edit help
  my_help git  [pull|push]    # git operations
  my_help hello               # hello
  my_help help [COMMAND]      # Describe available commands or one specific c...
  my_help init                # initialize my_help environment
  my_help list [HELP] [ITEM]  # list helps
  my_help new  [HELP]         # mk new HELP
  my_help set  [:key] [VAL]   # set editor or ext
  my_help version             # show version
...
```

例として my_help list と入力すると，

```
> my_help list
     org: - Emacs key bind
    todo: - my todo
   emacs: - Emacs key bind
...
```

様々なHELPの一覧が表示される. my_help list emacsと入力すると,

```
> my_help list emacs
my_help called with name : emacs, item :
-                  head :
-               license :
-          move cursor :
-            move page :
...
```

と表示され, my_help list emacs move cursorと入力すると,

```
>my_help list emacs move cursor
my_help called with name : emacs, item : move
item : move cursor
- c-f, move Forward,  Move to forward or right
- c-b, move Backward,      Move to back or left
- c-a, go Ahead of line,    Move to the beginning of the line
- c-e, go End of line,      Move to the end of the line
- c-n, move Next line,      Move to next line
- c-p, move Previous line,  Move to previous line
...
```

と出力される. これはemacsを使うときにコマンドによるカーソル操作を示している.

これによって瞬時にメモを引き出せるようにしている.

2.2 Test

本研究においては, TDD(Test Driven Development: テスト駆動開発) を規約として開発を進めた. テスト駆動開発とは、ソフトウェア開発の手法の一つで、プログラム本体より先にテストコードを書き、そのテストに通るように本体のコードを記述していく方式である [5].

2.2.1 RSpec

使用したテスティングフレームワークは RSpec であり. RSpec は次のような特徴がある. 当初 Testing framework は TDD を簡単に実行するために用意された. その後, 仕様 (spec) がアプリ開発の本質であるとすると, BDD(Behavior Briven Development: 振る舞い駆動開発) を実行する実装として, RSpec は開発された [6].

テストする RSpec の記述の一例を示す.

```
module MyHelp2
  RSpec.describe List do
    let(:templates_path) {
      File.join(File.dirname(__FILE__),
     "../../lib/templates")
    }
    describe "list の振る舞いで, " do
      it "ヘルプ名があるときは,
      その中の全ての item を表示" do
    output = "* hoge\n* hage"
    help_options = "example"
    expect(
  List.new(templates_path).
  list(help_options)
   ).to be_include(output)
...
```

RSpec は DSL(domain specific language) を使って，仕様を日本語で書くこと (describe)，その振る舞いを確かめること (expect().to be...) が記述できる．

このテストの結果は次のとおりとなる．

```
MyHelp2::List
 ...
  あいさつ
    あいさつを返す
  list の振る舞いで，
    ヘルプ名がないときは，全てのヘルプを表示
    ヘルプ名があるときは，その中の全ての item を表示
 ...
Finished in 0.01366 seconds (files took 0.43661 seconds to load)
5 examples, 0 failures
```

この出力結果は日本語通り，List を読み取ってヘルプ名があるかないかの違いで，振る舞いを確かめていることがわかる．前述のテストコードにあるとおり，'hoge, hage' は example.org というファイルの中身で，それを読み取って一致しているかを確かめている．

2.3 デザインパターン

本研究で実装しようとするデザインパターンである State, Strategy などについて概観しておく．

デザインパターンは, Gang of Four(GoF) と呼ばれる, Erich Ganma, Richard Helm, Ralph E. Johnson, John Matthew Vlissides[7] の 4 人によって 1995 年に出版されたカタログである [8]. Ruby の父と呼ばれるまつもとゆきひろはデザインパターンの意義を「しばしば使われる実用的なパターンに適切な名前を提供することによって, デザインにおける語彙を提供することにある」のだとしている [9, p.126].

このデザインパターンで提案されている 23 個のパターン中で，State あるいは Strategy と名付けられたパターンを本研究では利用する．このパターンが使用される状況を Martin

Fowler はリファクタリングの書籍の中で「タイプコードから State/Strategy へ」と名付けて紹介している [4, p.260]. 「クラスの振る舞いに影響を与え」メンテナンスしやすいコードに変更することを目的として，このパターンの適用手順を示している [4, p.73].

Ruby 独自の適用法として，Russ Olsen は「Ruby によるデザインパターン」の中で Strategy の使用法, Template からの導出, Proc 利用などを紹介している [10, p.60,p.71,p.77]. これらのテキストを参照しながら改良を加えていく.

第3章　結果

3.1　SelectHash の初期状態

3.1.1　org から md への移行

Emacs でマークアップ言語の org よりも．GUI である VSCode と Markdown の組み合わせが使われているため変更する必要がある．org から md に移行するために, case 文によって markup_select が md か org かによって挙動を変えるプログラムにしようとする．この挙動を実行する class として SelectHash を最初に切り出し select_hash.rb に置いた．その挙動を確認するため Test を select_hash_spec.rb に用意する．

3.1.2　select_hash_spec.rb の実装

select_hash は入力として md_text あるいは org_text を受け取り，markup_select に従って正しく hash を返す Method である．そこで，まずは，Hash が返っていることを確かめる Test(spec) を作る．

最初に assert を試してみる．

```
select_hash_spec.rb
    assert(contents1).to be(Hash)
```

しかし，結果は,

```
An error occurred while loading ./spec/select_hash_spec.rb.
Failure/Error: assert(contents1).to be(Hash)


NoMethodError:
  undefined method ‘assert’ for RSpec::ExampleGroups::SelectHash:Class
```

```
# ./spec/select_hash_spec.rb:15:in ‘block (2 levels) in <top (required)>’
```

とその Method がないという error が出た.

調べたところ，expect が使えるということだったので [11],

```
# select_hash_spec.rb
    expect(contents1).to be(Hash)

An error occurred while loading ./spec/select_hash_spec.rb.
Failure/Error: expect(contents1).to be(Hash)
   ‘expect‘ is not available on an example group (e.g. a ‘describe‘ or
   ‘context‘ block). It is only available from within individual examples
    (e.g. ‘it‘ blocks) or from constructs that run in the scope
    of an example (e.g. ‘before‘, ‘let‘, etc).
# ./spec/select_hash_spec.rb:15:in ‘block (2 levels) in <top (required)>’
```

となった.

このエラーは‘expect‘が example group(つまり describe などの block) の中では使えないという注意がされている．そこで，‘it‘ブロックに入れる.

この変更を加えた後では，エラーは

```
Failure/Error: expect(contents1).to be(Hash)

   expected #<Class:2140> => Hash
got #<SelectHash:2160> => #<SelectHash:0x000000011f1458c8>

   Compared using equal?, which compares object identity,
   but expected and actual are not the same object. Use
   ‘expect(actual).to eq(expected)‘ if you don’t care about
   object identity in this example.
```

となった．expected(期待値) が Hash であったのに対して，got(取得値) が SelectHash という object であることを注意している.

equal を使う比較が正しいかと問いかけてきている．つまり，equal は object の完全一致 (identity) を調べるため，expected と actual が完全には一致していないことを報告している．ここで，意図しているのは，Hash という一部であるかを調べることである．

参考文献によると，be_a などが使えるとわかった [11].

```
1) SelectHash hoge
    Failure/Error: expect(contents1).to be_a(Hash)
      expected #<SelectHash:0x000000012b981008> to be a kind of Hash
    # ./spec/select_hash_spec.rb:16:in `block (3 levels) in <top (required)>'
```

となり，正しく Hash を期待していることが示されている．

ここまでの code をまとめて記すと，

```
 1  it "正しく hash を返す" do
 2    [["md", md_text],
 3     ["org", org_text]].each do |markup_select, text|
 4      actual = SelectHash.new(markup_select, text) #.fsm_2
 5      expect(actual).to be_a(Hash)
 6      # contents1 = Md2hash.new(contents).fsm_2
 7
 8      #Print.new(contents1).list(1)
 9    end
10  end
```

1 行目の it ブロック内で expect を使えるように変更している．次に，SelectHash から戻り値を actual に入れている．これが Hash であるかを確認するために，5 行目で

```
expect the actual to be a Hash
```

と確認している．RSpec の構文はこの通り，英語の文章を書いた通りの code で記述されている．

この後は，期待通り Hash を返す code に SelectHash を変更していく．

3.1.3 select_hash.rb の改良

SelectHash の initialize では

```
def initialize(markup_select, text = "")
  # def initialize(context)
  case markup_select
    when "md"
      Md2Hash.new(text)
      #Md2Hash.new(context.text)
    when "org"
      Org2Hash.new(text)
      #Org2Hash.new(context.text)
    end
end
```

として markup_select に従って，Md2Hash あるいは Org2Hash を select している．

しかし，このままでは Md2Hash あるいは Org2Hash で読み込まれた text を Hash に直接保存できない．

そこで，新たに contents という method を作成し，これを外部から参照できるように変更する．

```
def initialize(markup_select, text = "")
  # def initialize(context)
  @contents = case markup_select
    when "md"
      Md2Hash.new(text).contents
      #Md2Hash.new(context.text)
    when "org"
      Org2Hash.new(text).contents
      #Org2Hash.new(context.text)
    end
```

```
end
```

こうすることで，SelectHash::contents として外部から参照できるようになる．そこで，select_hash_spec.rb の該当箇所を

```
actual = SelectHash.new().contents(markup_select, text)#.fsm_2
```

と変更することで，RSpec が

```
SelectHash
  正しく hash を返す

Finished in 0.00208 seconds (files took 0.10675 seconds to load)
1 example, 0 failures
```

と正しく通る事が確認できる．

3.2 Template Method パターンへの変更

これではコードが簡略化されていないので Template Method パターンを実装する．Template Method パターンとは Russ Olsen の記述 [10, p.60] に従って要約すると，

> Template Method パターンの一般的な考え方は，骨格となるメソッドを持った抽象基底クラスを構築することです．この骨格となるメソッド (テンプレートメソッドと呼ばれます) は抽象メソッドを呼ぶことによって変更に対応するような処理を扱います．抽象メソッドが呼び出されると，その実際の処理は具象サブクラスが提供することになります．様々な具象クラスの中から 1 つを選ぶことで，必要とする処理のバリエーションを選択することができます．

というものである．これを真似て，Org2Hash, Md2Hash の抽象クラスとなる Text2Hash を実装する．

```
class Text2Hash
  # abstract(抽象)
```

```
  attr_accessor :text, :contents, :results, :opts
  def initialize(context)
    #@opts = opts
    @text = context.split("\n")
    @opts = opts
    @contents = fsm
  end
  def fsm
    raise "Abstract method called"
  end
end
class Org2Hash < Text2Hash
  TRANS_ORG = {
    ...
  }
  def fsm
      ...
  end
end
class Md2Hash < Text2Hash
  TRANS_MD = {
    ...
  }
  def fsm
      ...
  end
end
```

となる．これにより Org2Hash, Md2Hash は Text2Hash を継承するように書き換え，一つのファイルに統合した.

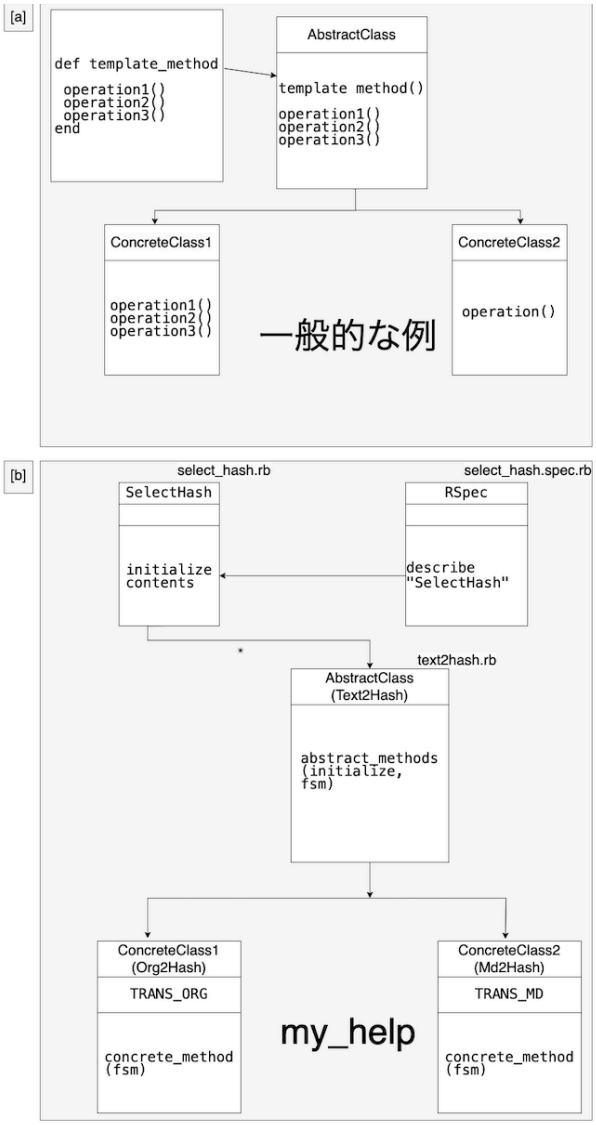

図 3.1: Template Method のクラス図

このコードを UML のクラス図で書くと図 3.1となる．Template Method の一般的な例が上パネルの [a]，my_help で実装したクラス図が下パネルの [b] である．一段目がクラスの名前, 二段目が変数の名前, 三段目が Method の名前である．まず, 一般的な例を説明すると, Template method が定義されており, そこには, operation1(), operation2(), operation3() の Method がある．これらの Method を Abstract class と呼ばれるクラスに渡し, そのクラスの同名の Methods に与える．それらの Method を具象クラス ConcreteClass1, ConcreteClass2 の 2 つのクラスに渡し, それぞれのクラスにある operation1(), operation2(), operation3() の Method に与える．Abstract class, ConcreteClass1, ConcreteClass2 は一般的な呼称である．

my_helpの方を説明すると, 先ほどのクラス図に当てはめたようなもので, select_hash.spec.rb のクラスは RSpec であり, そのクラスの Method は describe ”SelectHash”である．その Method を select_hash.rb に渡す． select_hash.rb のクラスは SelectHash であり，そのクラスの Method は initialize と contents である．それらの Method を text2hash.rb に渡す. Text2Hash は Abstract class であり, abstract_methods である initialize と fsm が Method である．それらの Method を ConcreteClass1 の Org2Hash, ConcreteClass2 の Md2Hash に渡す．Org2Hash には TRANS_ORG があり, concrete_method(fsm) がある．同様に Md2Hash には TRANS_Md があり, concrete_method(fsm) がある．なお, Text2Hash から渡される initialize はどちらも同じものである.

3.3 Stateへの変更

上述の code には case 文が含まれている．このような場合には，State パターンへ変更することが Martin Fawler の Refactoring のテキストでは推奨されている [4, p.300].

そこで紹介されている Refactoring 手順は以下の通りである.

1. 条件が大きなメソッドの一部になっている場合には，条件文を切り離し,「メソッドの抽出」(Extract Method) を行う.
2. 必要なら,「メソッドの移動」(Move Method) を使って，オブジェクト階層の適切な位置に条件文を移す.
3. ポリモーフィックなオブジェクトを 1 つ選ぶ．条件文のメソッドをオーバーライドするメソッドを書く．条件文の分岐先のコード本体を新メソッドにコピーし，調整する.
4. テストする.
5. 条件文のコピー済みの分岐を削除する.
6. テストする.
7. 条件文のすべての分岐先がポリモーフィックなメソッドに移されるまで，各分岐先について以上の作業を繰り返す.

この手順に従って変更を加えていく．まず，case 文になっている code を取り出すと次の通りである．

```
@contents = case markup_select
  when "md"
    Md2Hash.new(text).contents
  when "org"
    Org2Hash.new(text).contents
  end
```

元々，他のオブジェクトの属性に基づいて case 文を書くのは間違っている．case 文を使わなければならない場合には，他のオブジェクトではなく，自分自身のデータに基づいてそうすべきだ．

今，メソッドが使っている情報は，markup_select と text の二つである．この情報を SelectHash に置くのか，Md2Hash, Org2Hash に置くのかを検討しなければならない．今の場合は，contents メソッドの引数として markup_select が渡されているが，元来は initialize で渡されるべきである．text についても同じである．そこで，spec を元に戻して，initialize でこれらの引数を受け取り，クラス変数に保存する．test を作成している段階では，冗長な受け渡しであるが，大きな code 群の中では，分かれたところで定義される可能性がある．

3.4 Strategy への変更

しかし, Template Method パターンでは様々な問題点がある．これを Russ Olsen の記述 [10, p.71] に従って要約すると，

一つは

> このパターン (Template Method) が継承の上に成り立っているという事実からくるものです．. . . 継承に基づいた設計は，いくつかの深刻な不利益をもたらします．たとえどんなに注意深くコードを設計しても，サブクラスはそのスーパークラスに依存します．それは関連がもたらす当然の帰結なのです．

としています，

さらに

> Template Method パターンのような継承ベースのテクニックでは，実行時の柔軟性にも制限がついてしまいます．特定のバリエーションのアルゴリズムをひとたび選択すると, 方針を変えることが困難になります．Template Method パターンを使っていて，(レポートの) フォーマットを変更したくなったときは，全部を書き換えた (レポート) オブジェクトを作らなければなりません．単に，出力形式を切り替えたいがためだけに行うわけです.

として，Template Method では，少しの変更に大きなオブジェクトを作らなければならない不合理を説いている．

Russ Olsen の Strategy に関する記述は output に対する formatter であるが，現コードでは input に対する formatter の選択である．つまり，output の選択が html か plain text かであったのを，input の選択を org か md かに変更することであり，ほぼ同じ構造で変更すれば良い．

そこで，Russ Olsen も推薦する Strategy に変更していく．これは，

> 同じ目的を持った一群のオブジェクト, ストラテジ (Strategy) を定義することで, それぞれのストラテジオブジェクトは同じ仕事をこなすだけでなく, そのすべてが正確に同じインターフェイスを提供する.

というものである．

参考文献の実装例通り、formatter を実装する．まずは，formatter.rb にコピーした．

Template の abstract クラスであった Text2Hash はほぼそのまま残す．formatter を initialize の引数として受け取るように

```
class Text2Hash
  attr_accessor :text, :contents, :results, :formatter
  def initialize(context, formatter)
    @text = context.split("\n")
    @formatter = formatter
    @contents = input_text
  end
  ...
```

と変更する.

次に，今まで，Text2Hash の concrete クラスであった Org2Hash と Md2Hash クラスを Formatter クラスを継承するように変更する.（まずは）

そうすると

```
class Text2Hash
...
def input_text
  @formatter.input_text(@text)
  end
end

class Formatter
  def input_text(_text)
    raise 'Abstract method called'
  end
end

class Org2Hash < Formatter
...
```

となる.

1. input_text メソッドを Text2Hash クラスの中で作る
2. Formatter class は input_text の抽象メソッドとなる
3. Org2Hash(Md2Hash も) が Formatter を継承するようにする.
4. fsm メソッドの名前を input_text に変更し,
5. text を引数として受け取るように変更する.

さらに，select.rb の中の contents は

```
def contents
  case @markup_select
  when 'org'
    #      Org2Hash.new(@text,).contents
    Text2Hash.new(@text, Org2Hash.new).contents
  when 'md'
    #      Md2Hash.new(@text).contents
    Text2Hash.new(@text, Md2Hash.new).contents
  end
end
```

と変更されて，Org2Hash と Md2Hash という読み込むためのクラスを選択していたのが, 読み込むための format を選択するように明示的に呼び出されるようになる.

このコードを UML のクラス図で書くと図 3.2となる．Strategy の一般的な例が上パネルの [c],my_help で実装したクラス図が下パネルの [d] である．先程の例と同様, 一段目がクラスの名前, 二段目が変数の名前, 三段目が Method の名前である．まず, 一般的な例を説明すると, Context が定義されており, そこには, @strategy という Method がある．この Method を Strategy と呼ばれるものに渡し, operation() という Method に与える．その Method を, Strategy1, Strategy2 の 2 つのクラスに渡し, それぞれのクラスにある operation() の Method に与える.

my_help の方を説明すると, 先ほどのクラス図に当てはめたようなもので, select_hash.spec.rb のクラスは RSpec であり，そのクラスの Method は describe ”SelectHash”である． その Method を select_hash.rb に渡す． select_hash.rb のクラスは SelectHash であり，そのクラスの Method は initialize と contents である．それらの Method を formatter.rb に渡す．Formatter は Strategy class であり, Strategy_methods である initialize と input_text が Method である．それらの Method を Strategy1 の Org2Hash, Strategy2 の Md2Hash に渡す．Org2Hash には TRANS_ORG があり, Strategy_method(input_text) がある．同様に Md2Hash には TRANS_MD があり, Strategy_method(input_text) がある．なお, Formatter から渡される initialize はどちらも同じものである.

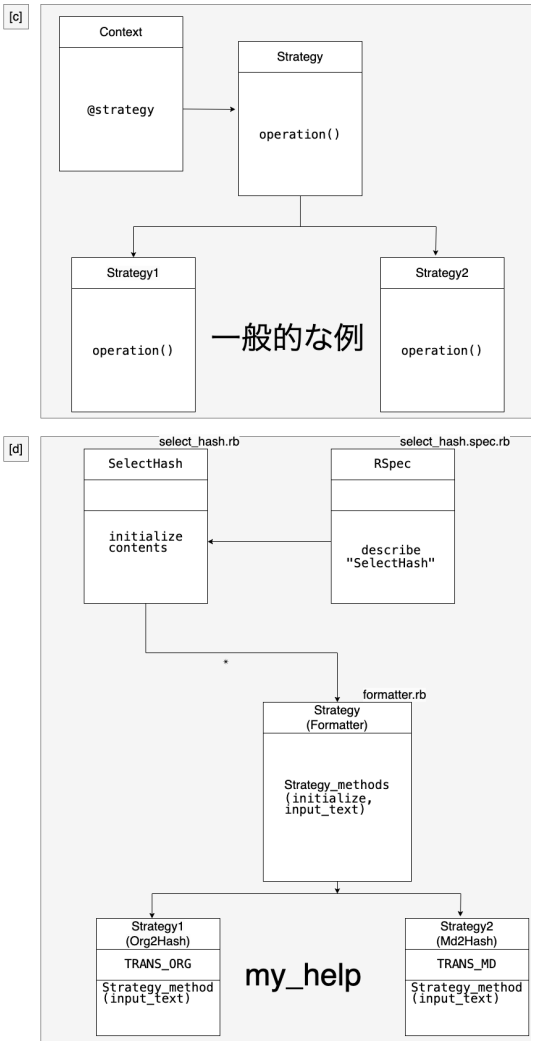

図 3.2: Strategy のクラス図

3.5 proc化

Ruby では Strategy は proc 化によって簡略化することが可能になる．Russ Olsen の記述 [10, p.77] に従って要約すると， 以下の通りになる．

> 私たちがオブジェクト指向プログラミング言語を使っているときは, オブジェクトとオブジェクト間の関係について考えることに多くの時間を割きます. しかし, コードのことについてはあまり深く考えない傾向があります. オブジェクトからデータを分離することは造作もないことです. @text をレポートオブジェクトから取り出し, ほかに依存せずに取り回すことはできます. しかしながら私たちは, オブジェクトと, そこに記述してあるコードというのは, 強く結合していて分離しがたいものと考える傾向があります. もちろんそうでなけれ

ばならないということはありません. ひとかたまりのコードをオブジェクトから取り出し，オブジェクトのように取り回せたらどうでしょう? Ruby はまさにそれができるのです. Ruby では Proc オブジェクトはコードのかたまりを保持するオブジェクトです.

参考文献を参照して proc オブジェクトを作ると, selct_hash.rb の代わりになる, select_hash_proc.rb の実装する. それに対応し, select_hash_spec.rb の代わりになる select_hash_proc_spec.rb の実装もする.

select_hash_proc_spec.rb は select_hash_spec.rb の 2 行目にある, RSpec.describe 'SelectHash' do を RSpec.describe 'SelectHashProc' do と変更したものである. select_hash_proc.rb に formatter.rb のコードを入れる.

```
class Formatter
  def initialize(head_mark)
    @line_match = /^#{head_mark} (.+)/
    @transfer_hash = mk_transfer_hash(head_mark)
  end
```

Formatter に initialize を入れ, そこに line.match と @transfer_hash を入れている. head_mark で受け渡しをするものとなっている.

そのコードに対応させると select_hash.rb にあった

```
class SelectHashProc
  def initialize(markup_select, text = '')
    @markup_select = markup_select
    @text = text
  end


  def contents
    case @markup_select
    when 'org'
      head_mark = '*'
```

```
      Text2Hash.new(@text, Formatter.new(head_mark)).contents
    when 'md'
      head_mark = '#'
      Text2Hash.new(@text, Formatter.new(head_mark)).contents
    end
  end
end
```

と変える. "*" と"#" で判別できるため, TRANS_ORG, TRANS_MDが要らなくなり,

```
def mk_transfer_hash(head_mark)
    return {
      # current
      #            new         action
      #-----------------------------------
      init: {
:default => %i[init ignore],
head_mark => %i[reading item]
      },
      reading: {
head_mark => %i[reading item],
:default => %i[reading data]
      }
    }
  end
```

となる. 結果, コードを大幅に減らすことができた.

第4章　まとめ

本研究では，my_help を md に対応するために行なった．この対応のために以下の変更を行なった.

- Template Method の導入
- State への変更
- Strategy への変更
- proc 化

これらの改良を加えたことで，これまで org のみしか my_help は利用できなかったが, 拡張子を ”.md ”と指定することで Markdown で記述した内容も利用することが可能となった.

また，Strategy への変更, proc 化によってプログラムのコードを省略し, 改良しやすいものとなったと考えている.

謝辞

終始熱心なご指導とご助言を賜りました西谷滋人教授に感謝の意を表します．教授のご指導のおかげで, Ruby を理解し研究を行うことができました．そして, 研究の進行に伴い, 西谷研究室の同輩や，先輩方に様々な助力や助言，知識の供給を頂きました．心より感謝いたします．本当にありがとうございました．

参考文献

[1] ”my_help の階層表示の作成”, 佐々木陸,(関西学院大学卒業論文 2022).

[2] ”my_help/README.org”, Daddygongon, https://github.com/daddygongon/my_help.

[3] ”Ruby の gem とは?概要から使い方までの解説まとめ”, TECH PLAY, https://techplay.jp/column/529.

[4] ”リファクタリング:Ruby エディション”, Jay Fields, Shane Harvie, and Martin Fowler, (ASCII MEDIA WORKS 2010).

[5] ”IT 用語辞典 E WORDS” IT 用語辞典 E WORDS, https://e-words.jp/w/テスト駆動開発.html.

[6] ”Effective Testing with RSpec 3”, M.Marston and E.Dees, (Pragmatic Bookshelf, 2017).

[7] ”G学院プログラミング用語解説”, G学院, https://gimo.jp/glossary/details/gang_of_four.html.

[8] ”オブジェクト指向における再利用のためのデザインパターン”, Erich Gamma, Ralph Johnson, Richard Helm, and John Vlissides, (ソフトバンククリエイティブ, 1995).

[9] ”まつもとゆきひろ コードの世界 スーパー・プログラマーになる 14 の思考法”, まつもとゆきひろ, (日経 BP 2009).

[10] ”Ruby によるデザインパターン”, Russ Olsen, (ピアソン・エデュケーション 2009).

[11] ”Type matchers”, relish, https://relishapp.com/rspec/rspec-expectations/docs/built-in-matchers/type-matchers.