2019年7月13日

Ruby初級者向けレッスン 71回
― テスト駆動開発 ―

ひがき @ Ruby関西

お品書き

- テスト フレームワーク いろいろ
- テスト駆動開発
  - Red–Green–Refactor
- Test Double

テスト フレームワーク いろいろ

- Test::Unit
- RSpec
- Cucumber
- etc.

Test::Unit

- プログラムの単体テストを行う
- Ruby用の xUnit
  - MiniTest::Unit — 標準添付
  - test-unit Gem — bundled gem
- The history of testing framework in Ruby

  http://slide.rabbit-shocker.org/authors/kou/rubykaigi-2015/

RSpec

- プログラムの振舞いを記述する (BDD)
- ドメイン特化言語 (DSL)
- スはスペックのス by 角谷さん、諸橋さん

  http://jp.rubyist.net/magazine/?0021-Rspec

Cucumber (::)

- 自然言語に近い形式で書ける
- 受け入れテスト向け
- ……自動化は Cucumber から Turnip へ by 福井さん

  http://magazine.rubyist.net/
  ?0042-FromCucumberToTurnip

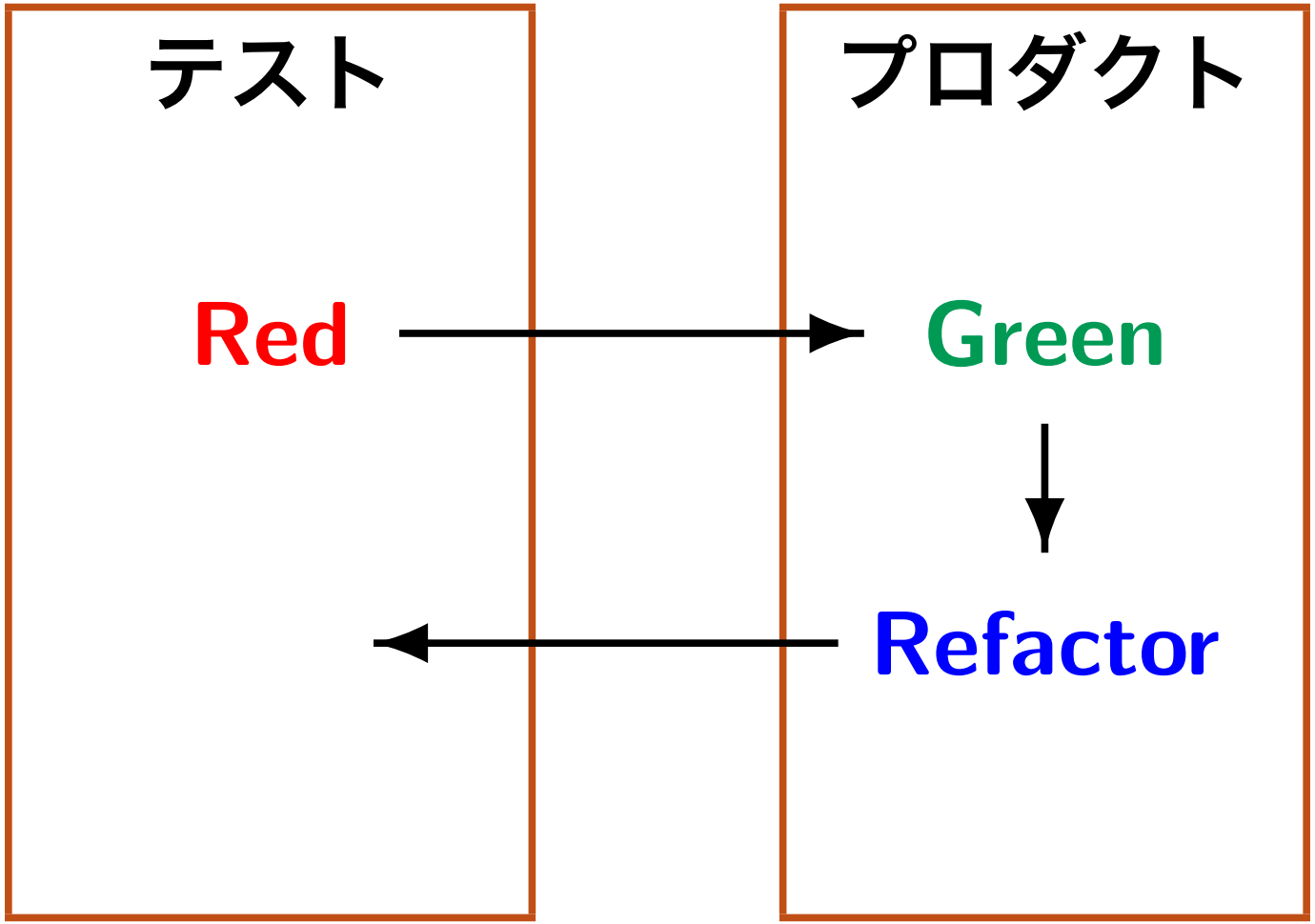
テストのメリット

- 機能的な保証
  - バグの検出
  - バグ混入を監視
- 開発支援
  - 不安を解消
  - リファクタリング

テスト駆動開発

1. まずテストを書き、失敗することを確認する
2. テストが成功するように、製品コードを書く
3. テストはそのまま、コードをきれいに

Red–Green–Refactor



テスト駆動開発をやってみよう

- プロダクトコード — `myarray.rb`
- テストコード — `test_myarray.rb`

- MyArray が空のとき

  `MyArray#empty?` は真であるべき

Step 0 プロダクト

```
class MyArray
  def initialize
    @array = []
  end
end
```

Step 0 テスト

```
require 'test/unit'
require_relative 'myarray'
class TestMyArray < Test::Unit::TestCase
  def setup
    @array = MyArray.new
  end
  def test_empty_by_empty_myarray
    assert @array.empty?
  end
end
```

テストのながれ

テストごとに、以下を実行する。

1. setup — 準備
2. test_〜 — テスト本体
3. teardown — 後始末

- テストの実行順序は無保証
- 各テストは他のテストに依存してはいけない

Step 0 ERROR 実行

```
$ ruby test_myarray.rb
    ...
E
================================================================
Error: test_empty_by_empty_myarray(TestMyArray)
  NoMethodError: undefined method `empty?' for
 #<MyArray:0x802ea72c @array=[]>
test_myarray.rb:12:in `test_empty_by_empty_myarray'
    ...
1 tests, 0 assertions, 0 failures, 1 errors,
 0 pendings, 0 omissions, 0 notifications
```

Step 1 プロダクト

```
class MyArray
  def initialize
    @array = []
  end

  def empty?      # <= 追加
  end             # <= 追加
end
```

Step 1 Red 実行

```
$ ruby test_myarray.rb
    ...
F
==========================================================
Failure: <nil> is not true.
test_empty_by_empty_myarray(TestMyArray)
test_myarray.rb:12:in ‘test_empty_by_empty_myarray’
    ...
1 tests, 1 assertions, 1 failures, 0 errors,
 0 pendings, 0 omissions, 0 notifications
```

Step 2 プロダクト

```
class MyArray
  def initialize
    @array = []
  end

  def empty?
    true            # <= 追加
  end
end
```

Step 2 Fake it! 実行

```
$ ruby test_myarray.rb
Loaded suite test_myarray
Started
.

Finished in 0.000942 seconds.

1 tests, 1 assertions, 0 failures, 0 errors,
 0 pendings, 0 omissions, 0 notifications
100% passed
```

assert

式が真なら、テストが通る。

```
assert("".empty?, "空であること")
assert(obj.nil?,  "nil であること")
```

assert_equal

第1引数 == 第2引数 なら、テストが通る。

```
assert_equal(0, array.size)
assert_equal('matz', author)
assert_equal(0.3, 0.2 + 0.1, "Fail!!")
```

```
Fail!!
<0.3> expected but was
<0.30000000000000004>
```

assert_in_epsilon

第1引数と第2引数の誤差が
　　第3引数の許容範囲内なら、テストが通る。

```
assert_in_epsilon(0.3, 0.2 + 0.1, 1e-15)
```

assert_raise

ブロック内で、第1引数の例外が発生したら、
テストが通る。

```
ex = assert_raise(IndexError) do
  [].fetch(0)
end
```

Step 3 プロダクト

```
class MyArray
    ...
  def empty?
    true              # <= Fake it!
  end

  def << o            # <= 追加
    @array << o       # <= 追加
    self              # <= 追加
  end                 # <= 追加
end
```

Step 3 テスト

```
    ...
class TestMyArray < Test::Unit::TestCase
    ...
  def test_empty_by_empty_myarray
    assert @array.empty?
  end
  def test_empty_by_nonempty_myarray  # <= 追加
    @array << "1st"                   # <= 追加
    refute @array.empty?              # <= 追加
  end                                 # <= 追加
end
```

Step 3 Triangulate 実行

```
$ ruby test_myarray.rb
    ...
.F
===============================================
Failure: test_empty_by_nonempty_myarray(TestMyArray)
<true> is neither nil or false.
  ...
2 tests, 2 assertions, 1 failures, 0 errors,
 0 pendings, 0 omissions, 0 notifications
```

Step 4 プロダクト

```
class MyArray
    ...
  def empty?
    @array.empty?    # <= 修正
  end

  def << o
    @array << o
    self
  end
end
```

Step 4 Green 実行

```
$ ruby test_myarray.rb
Loaded suite test_myarray
Started
..

Finished in 0.00123 seconds.

2 tests, 2 assertions, 0 failures, 0 errors,
 0 pendings, 0 omissions, 0 notifications
100% passed
```

こんなときどうする

```
require 'person'
class TestPerson < Test::Unit::TestCase
  def test_age
    matz = Person.new(
        'matz', Time.local(1965, 4, 14))

    assert_equal 54, matz.age
    # 来年はテストに失敗する!?
  end
end
```

Test Double

- テスト対象 (プロダクト) が依存している
  コンポーネントを置き換える
  - ダミーオブジェクト
  - テストスタブ
  - テストスパイ
  - モックオブジェクト
  - フェイクオブジェクト

http://goyoki.hatenablog.com/entry/
20120301/1330608789

例えば影武者を使う

```
require 'kagemusha/time'
class TestPerson < Test::Unit::TestCase
  def test_age
    matz = Person.new(
        'matz', Time.local(1965, 4, 14))

    Kagemusha::Time.at(2019, 7, 13) do
      assert_equal 54, matz.age
    end
  end
end
```

もしくは Mocha を使う

```
require 'mocha/setup'
class TestPerson < Test::Unit::TestCase
  def test_age
    matz = Person.new(
        'matz', Time.local(1965, 4, 14))

    Time.stubs(:now)
        .returns(Time.local(2019, 7, 13))
    assert_equal 54, matz.age
  end
end
```

演習問題 0

今日のレッスンで分からなかったこと、疑問に思ったことをグループで話し合ってみよう。

演習問題 1

- スタッククラスを作ろう!
  - テスト駆動開発で「スタック」クラスを作ってみよう。
  - push で順に要素を追加できること。
  - pop で最後から順に要素を取り出せること。
  - このような動作を LIFO (Last In, First Out) という。

実装するメソッド

- `empty?`
  - スタックが空なら真、空でなければ偽を返す。
- `size`
  - スタックの要素数を返す。
- `push(val)`
  - 引数の値をスタックの最後に積む。
- `pop`
  - スタックの最後の値を取り除いて返す。
  - スタックが空なら`Stack::EmptyStackError`を発生させる。

Step 1 empty?

- 新しいスタックの empty? は真であること。
- テストが失敗することを確認しよう。**Red**
- empty? で真を返そう。**Fake it!**

Step 2 empty?

- push を実装して空でないスタックを作ろう。
  空でないスタックの empty? は偽であること。
- Fake it! が通らなくなることを確認しよう。
  Triangulate
- テストが通るように empty? を実装しよう。
  Green

Step 3 `size`

- 新しいスタックの `size` は 0 であること。
- テストが失敗することを確認しよう。**Red**
- `size` で 0 を返そう。**Fake it!**

Step 4 `size`

- 要素がひとつ積まれたスタックの `size` は 1 であること。
- Fake it! が通らなくなることを確認しよう。

  Triangulate
- テストが通るように `size` を実装しよう。

  Green

Step 5 pop

- 要素がふたつ (例えば :ALFA と :BRAVO が) 積まれたスタックから `pop` すると最後の要素 (:BRAVO) が返ること。
- `size` は 1 であること。
- `empty?` は偽であること。

Step 6 pop

- 要素がひとつ (例えば “Alice” が) 積まれたスタックから `pop` すると最後の要素が返ること。
- `size` は 0 であること。
- `empty?` は真であること。

Step 7 pop

- 新しいスタックから pop すると
  Stack::EmptyStackError が発生すること。

自己紹介

- 名前 (ニックネーム)
- 普段の仕事・研究内容・代表作
- Ruby歴・コンピュータ歴
- 勉強会に来た目的
- などなど

参考

- **るりま**

  https://docs.ruby-lang.org/ja/
- **Test::Unit リファレンスマニュアル**

  https://test-unit.github.io/test-unit/ja
- **サンプルコード**

  https://github.com/higaki/
  learn_ruby_kansai_87