Ruby入門

- 第９章　例外処理を理解する -

P329 9.2 - P340 9.3

DIVE INTO CODE

もくじ

9.2.1　発生した例外を捕捉しない場合

workspace > ruby-book > test > test.rb > ...

```
1  puts 'Start.'
2  module Greeter
3      def hello
4          'hello'
5      end
6  end
7  greeter = Greeter.new
8  puts 'End.'
9
```

モジュールはインスタンスを作成することができないので例外（エラー）が発生。
Rubyコマンドで実行した場合は、その後の処理「puts 'End.'」は実行されない。

例外は通常、英語でメッセージが表示されます。
「未定義のメソッドであるnewをGreeterモジュールに対して呼び出そうとした」

問題　出力　…　Code

```
[Running] ruby "/Users/sakamoto/workspace/ruby-book/test/test.rb"
/Users/sakamoto/workspace/ruby-book/test/test.rb:7:in `<main>':
undefined method `new' for Greeter:Module (NoMethodError)
Start.
```

irbで実行した場合、以下が表示される。
この情報をバックトレースやスタックトレースと呼ぶ。
内容の読み方は第11章(デバック技法)で説明。

```
Traceback (most recent call last):
        4: from /Users/sakamoto/.rbenv/versions/2.6.5/bin/irb:23:in `<main>'
        3: from /Users/sakamoto/.rbenv/versions/2.6.5/bin/irb:23:in `load'
        2: from /Users/sakamoto/.rbenv/versions/2.6.5/lib/ruby/gems/2.6.0/gems/irb-1.0.0/exe/irb:11:in `<top (required)>'
        1: from (irb):1
NoMethodError (undefined method `ruby' for main:Object)
```

9.2.2　例外を捕捉して処理を続行する場合

workspace > ruby-book > test > test.rb > { } Greeter > hello

```
1   puts 'Start.'
2   module Greeter
3     def hello
4       'hello'
5     end
6   end
7
8   begin
9     # 例外が起きうる処理
10    greeter = Greeter.new
11  rescue
12    # 例外が発生した場合の処理
13    puts '例外が発生したが、このまま続行する'
14  end
15
16  puts 'End.'
17
```

この書き方は例外処理のもっとも単純な構文。
実際のプログラムではこのようなコードは好ましくない。例外処理の良し悪しについてはのちほど。

問題　出力　デバッグ コンソール　ターミナル

```
[Running] ruby "/Users/sakamoto/workspace/ruby-
Start.
例外が発生したが、このまま続行する
End.
```

例外を捕捉して処理を続行したことで
途中でエラーになることなく処理が完了した。

9.2.3　例外処理の流れ

workspace > ruby-book > test > test.rb > method_3

```ruby
def method_1
  puts '元気ですかー！！'
  begin
    method_2
  rescue
    puts '例外が発生しました'
  end
  puts '3 ダー！！！！'
end

def method_2
  puts '元気があれば何でもできる'
  method_3
  puts '2'
end

def method_3
  puts 'いくぞー！！'
  1/0  # ZeroDivisionErrorを発生させる（整数の 0 で除算を行ったときに発生）
  puts '1'
end

method_1
```

例外発生！！

問題　出力　デバッグ コンソール　ターミナル

```
[Running] ruby "/Users/sakamoto/workspace/ruby-book/test/test.rb"
元気ですかー！！
元気があれば何でもできる
いくぞー！！
例外が発生しました
3 ダー！！！！
```

このコードの例外処理について

① method_3の途中で例外を発生！！

② 例外の捕捉が無いため、method_2へ戻る

③ 例外の捕捉が無いため、method_1へ戻る

④ rescueの処理

⑤ method_1の処理が続行し正常終了する

もし、method_1にも例外処理がなかった場合

上記①～③まで同じで、method_1でも例外の捕捉がされずに
エラー(ZeroDivisionError)の異常終了になる

9.2.4　例外オブジェクトから情報を取得する

workspace > ruby-book > test > test.rb

```ruby
begin
  1/0 #例外が起きうる処理
rescue => e #例外オブジェクトを格納する変数
  # 例外が発生した場合の処理
  puts "エラークラス: #{e.class}"
  puts "エラーメッセージ: #{e.message}"
  puts "バックトレース------"
  puts e.backtrace
  puts "--------"
end

```

問題　出力　デバッグ コンソール　ターミナル

```
[Running] ruby "/Users/sakamoto/workspace/ruby-book/test/test.rb"
エラークラス: ZeroDivisionError
エラーメッセージ: divided by 0
バックトレース------
/Users/sakamoto/workspace/ruby-book/test/test.rb:2:in `/'
/Users/sakamoto/workspace/ruby-book/test/test.rb:2:in `<main>'
--------
```

例外オブジェクトの情報を取得するためのメソッド紹介

class : エラーのクラスを返す

message : エラーのメッセージを返す

backtrace : エラーのバックトレースを返す

9.2.5　クラスを指定して補足する例外を限定する

```ruby
begin
  1/0
rescue ZeroDivisionError #捕捉したい例外クラス
  puts '0で除算しました'
end
```

例外クラスを指定して例外処理ができる

```
[Running] ruby "/Users/sakamoto/workspace/ruby-book/test/test.rb"
0で除算しました
```

```ruby
begin
  'abc'.hoge #NoMethodErrorを発生させる
rescue ZeroDivisionError #捕捉したい例外クラス
  puts '0で除算しました'
end
```

ZeroDivisionError以外は例外が捕捉されない

```
[Running] ruby "/Users/sakamoto/workspace/ruby-book/test/test.rb"
/Users/sakamoto/workspace/ruby-book/test/test.rb:2:in `<main>':
undefined method `hoge' for "abc":String (NoMethodError)
```

```ruby
begin
  'abc'.hoge #NoMethodErrorを発生させる
rescue ZeroDivisionError #捕捉したい例外クラス
  puts '0で除算しました'
rescue NoMethodError #捕捉したい例外クラス
  puts '存在しないメソッドが呼び出されました'
end
```

複数のrescueで例外クラスを指定できる

```
[Running] ruby "/Users/sakamoto/workspace/ruby-book/test/test.rb"
存在しないメソッドが呼び出されました
```

```ruby
begin
  'abc'.hoge #NoMethodErrorを発生させる
rescue ZeroDivisionError, NoMethodError
  puts '0で除算したか、存在しないメソッドが呼び出されました'
end
```

1 つのrescueに例外クラスを複数指定できる

```
[Running] ruby "/Users/sakamoto/workspace/ruby-book/test/test.rb"
0で除算したか、存在しないメソッドが呼び出されました
```

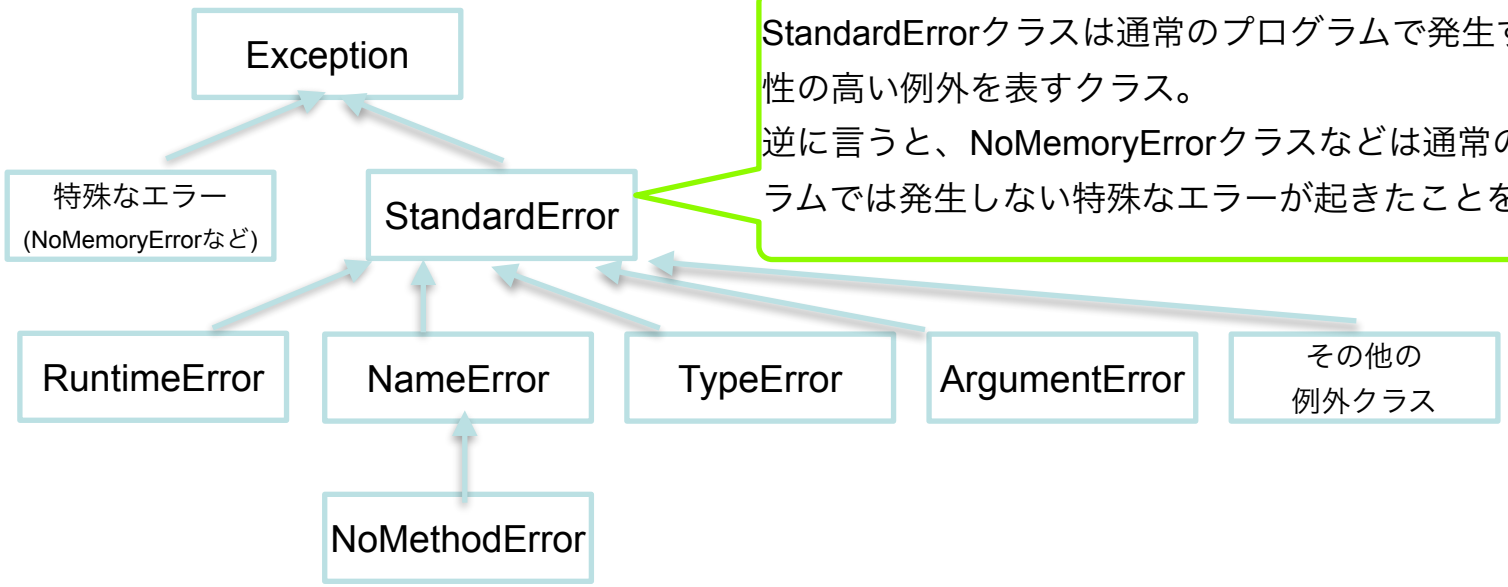
workspace > ruby-book > test > test.rb

```ruby
begin
  'abc'.hoge #NoMethodErrorを発生させる
rescue ZeroDivisionError, NoMethodError => e
  puts '0で除算したか、存在しないメソッドが呼び出されました'
  puts "エラー:#{e.class} #{e.message}"
end
```

例外オブジェクトを変数に格納することができる

```
[Running] ruby "/Users/sakamoto/workspace/ruby-book/test/test.rb"
0で除算したか、存在しないメソッドが呼び出されました
エラー:NoMethodError undefined method `hoge' for "abc":String
```

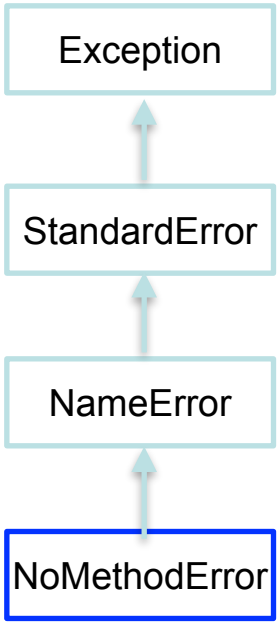
9.2.6　例外クラスの継承関係を理解する




```
1  # 例外処理の悪い書き方
2  begin
3    # 例外が起きそうな処理
4  rescue Exception
5    # Exceptionとそのサブクラスが捕捉される。
6    # つまりNoMemoryErrorなども捕捉される。
7  end
```

通常のプログラムで捕捉するのはStandardErrorクラスか、
そのサブクラスに限定すべき！
Exceptionクラスや、StandardErrorと無関係の例外クラスを
指定することは避けましょう。

9.2.7　継承関係とrescue節の順番に注意する




```ruby
begin
  'abc'.hoge #NoMethodErrorを発生させる
rescue NameError
  # NoMethodErrorはここで捕捉される
  puts 'NameErrorです'
rescue NoMethodError
  # このrescue節は永遠に実行されない
  puts 'NoMethodErrorです'
end
```

```
問題　出力　デバッグ コンソール　ターミナル
[Running] ruby "/Users/sakamoto/workspace/ruby-
NameErrorです
```

スーパークラスのNameErrorを先に書いてしまうと、
２つめのrescue節（サブクラスのNoMeethodError）
に到達することができない

```ruby
begin
  'abc'.hoge #NoMethodErrorを発生させる
rescue NoMethodError
  # NoMethodErrorはここで捕捉される
  puts 'NoMethodErrorです'
rescue NameError
  puts 'NameErrorです'
end
```

```
問題　出力　デバッグ コンソール　ターミナル
[Running] ruby "/Users/sakamoto/workspace/rub
NoMethodErrorです
```

【解決法】
スーパークラスよりもサブクラスを手前に
もってくるようにすればよい

このように例外処理を書く場合は
例外クラスの継承関係を意識する！！
※細かいテクニックはP339の下の方を参照

9.2.8　例外発生時にもう一度処理をやりなおすretry

```ruby
begin
  # 例外が発生するかもしれない処理
rescue
  retry # 処理を繰り返す
end
```

ネットワークエラーのように一時的に発生している問題が例外の原因であれば、
何度かやり直すことで正常に実行出来る可能性があります。
そんな場合は**retry**を使うことで、begin節の最初からやり直せます。

ただし無条件にretryし続けると、例外が解決しない場合に**無限ループ**になってしまうので、retryの回数を制限すると良いです。