RSpec Mocks

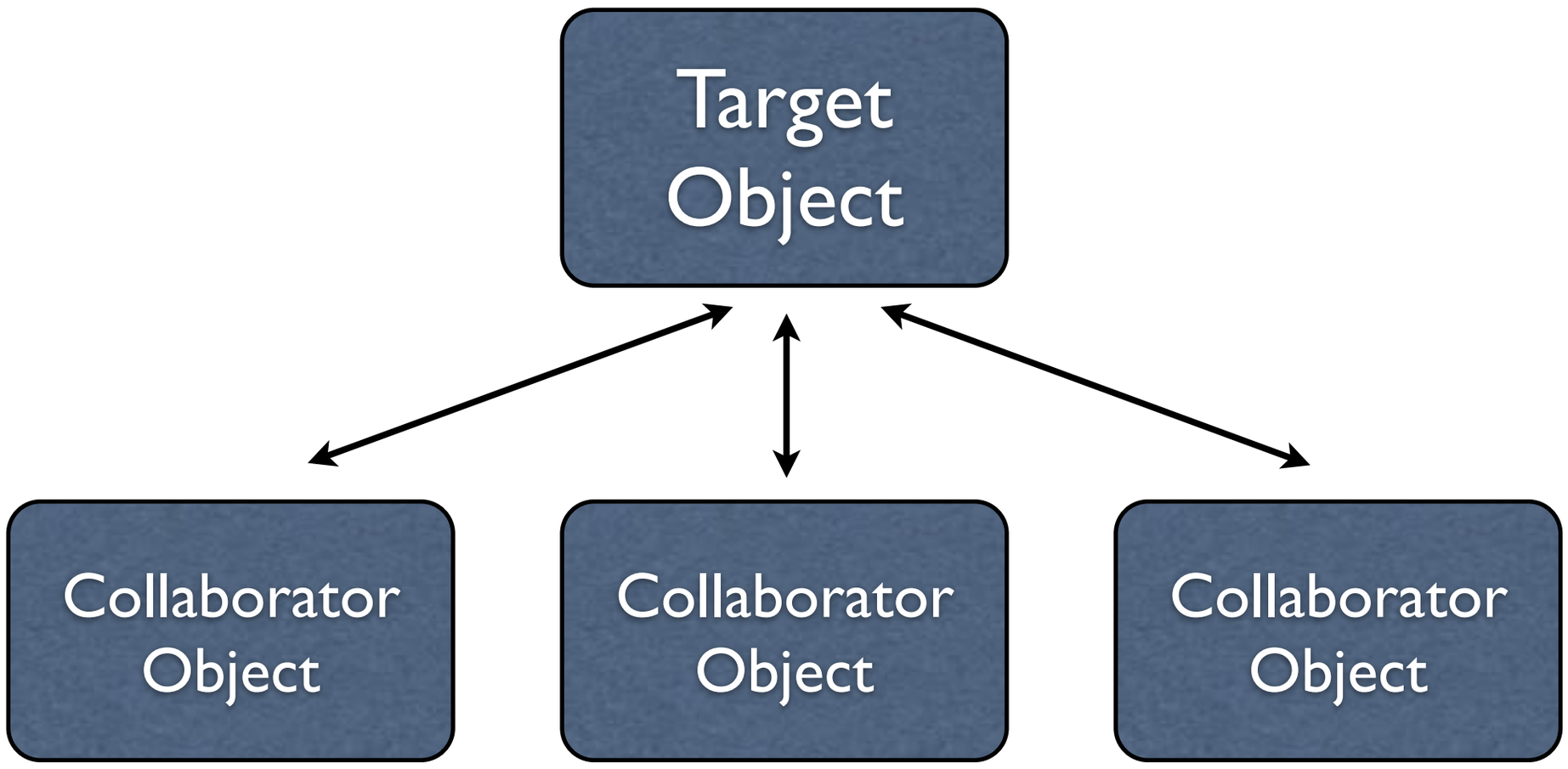
https://ihower.tw

2016/8

RSpec Mocks

用假的物件替換真正的物件，作為測試之用

物件導向是


Target Object
Collaborator Object
Collaborator Object
Collaborator Object

當你在寫目標類別的測試和實作時，這些 Collaborator 可能...

- 無法控制回傳值的外部系統 (例如第三方 web service)
- 建構正確的回傳值很麻煩 (例如得準備很多假資料)
- 可能很慢，拖慢測試速度 (例如耗時的運算)
- 有難以預測的回傳值 (例如亂數方法)
- 還沒開始實作 (特別是採用 Pure TDD 流程)

使用假物件

- 可以隔離 Collaborator 的 Dependencies
  - 讓你專心在目標類別上
  - 只要 Collaborator 提供的介面不變，修改實作不會影響這邊的測試。

用 double 產生假物件

```
@user = double("user", :name => "ihower")
@user.name # "ihower"
```

```
@customer = double("customer").as_null_object
@customer.foobar # nil
```

測試狀態

```
describe "#receiver_name" do
  it "should be user name" do
    user = double(:user, :name => "ihower")
    order = Order.new(:user => user)
    expect(order.receiver_name).to eq("ihower")
  end
end
```

測試行為 Mock

如果沒被呼叫到，就算測試失敗

```
@gateway = double("ezcat")

# 預期等會 @gateway 必須被呼叫到 deliver 方法
expect(@gateway).to receive(:deliver).with(@user).and_return(true)

# 發生行為，如果沒有就測試失敗
```

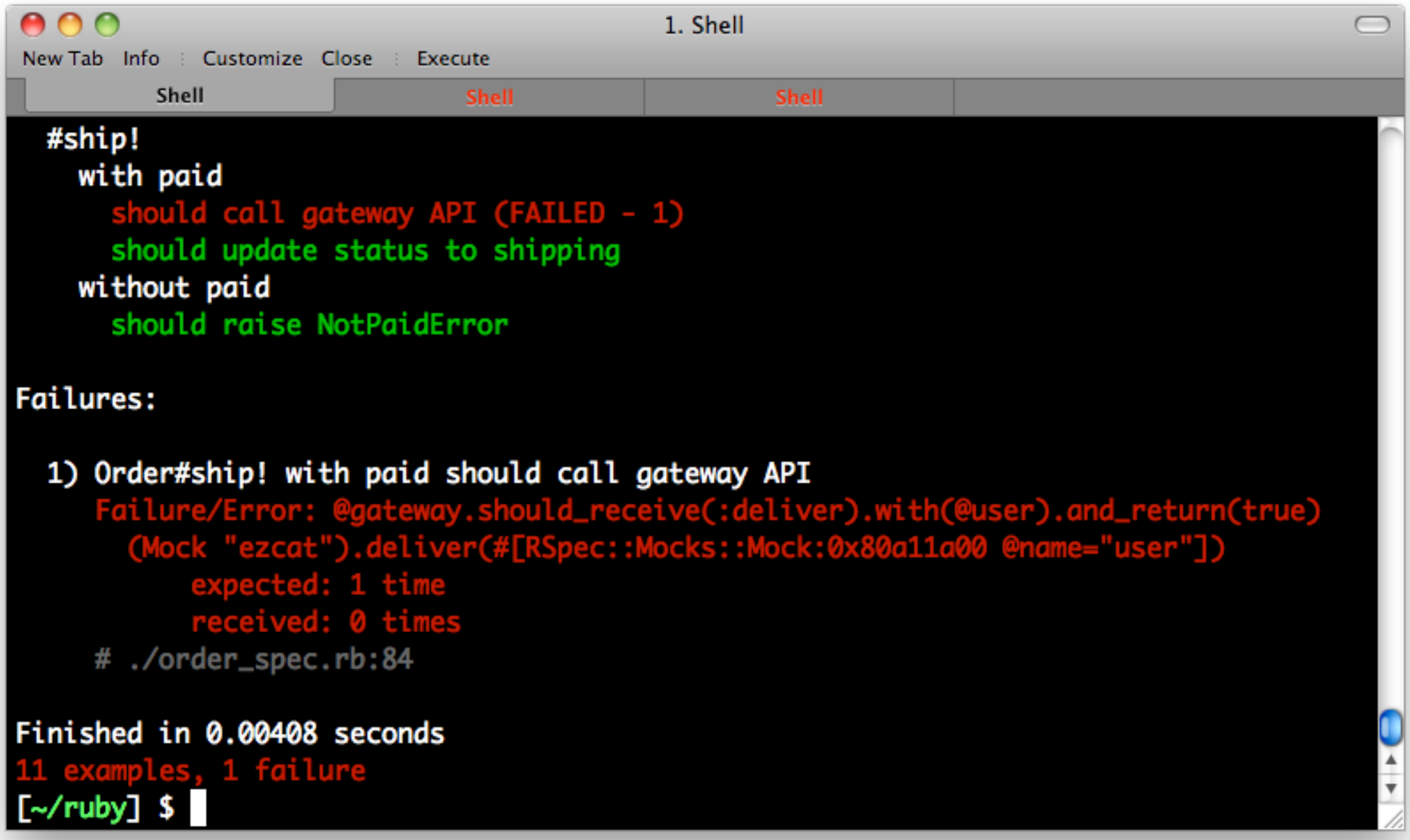
用來測試行為的發生

```
describe "#ship!" do

  before do
    @user = double("user").as_null_object
    @gateway = double("ezcat")
    @order = Order.new( :user => @user, :gateway => @gateway )
  end

  context "with paid" do
    it "should call ezship API" do
      expect(@gateway).to receive(:deliver).with(@user).and_return(true)
      @order.ship!
    end
  end
```

行為沒發生...


1. Shell
New Tab Info Customize Close Execute
Shell
Shell
Shell
#ship!
with paid
should call gateway API (FAILED - 1)
should update status to shipping
without paid
should raise NotPaidError
Failures:
1) Order#ship! with paid should call gateway API
Failure/Error: @gateway.should_receive(:deliver).with(@user).and_return(true)
(Mock "ezcat").deliver(#[RSpec::Mocks::Mock:0x80a11a00 @name="user"])
expected: 1 time
received: 0 times
./order_spec.rb:84
Finished in 0.00408 seconds
11 examples, 1 failure
[~/ruby] $

Partial mocking and stubbing

- 如果 Collaborator 類別已經有了，我們可以 reuse 它
- 只在有需要的時候 stub 或 mock 特定方法

Partial Stub

可以用在任意物件及類別上，假造他的方法和回傳值

```
user = User.new
allow(user).to receive(:find).and_return("ihower")
```

Partial Mock

可以用在任意物件及類別上，假造他的方法和回傳值，
並且測試他必須被呼叫到

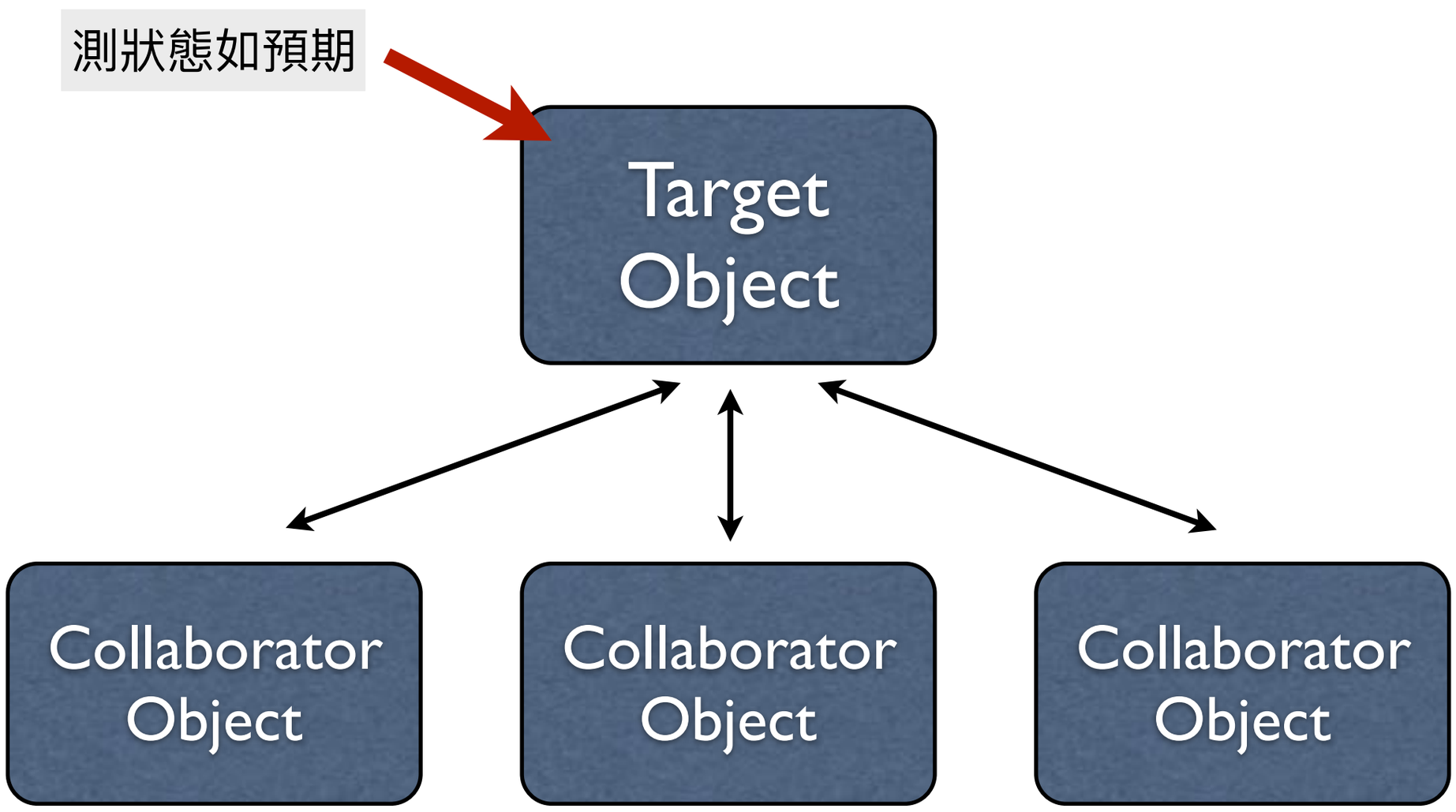
```
gateway = Gateway.new
expect(gateway).to receive(:deliver).with(user).and_return(true)
```

一般測試流程 vs.
Mock 測試流程

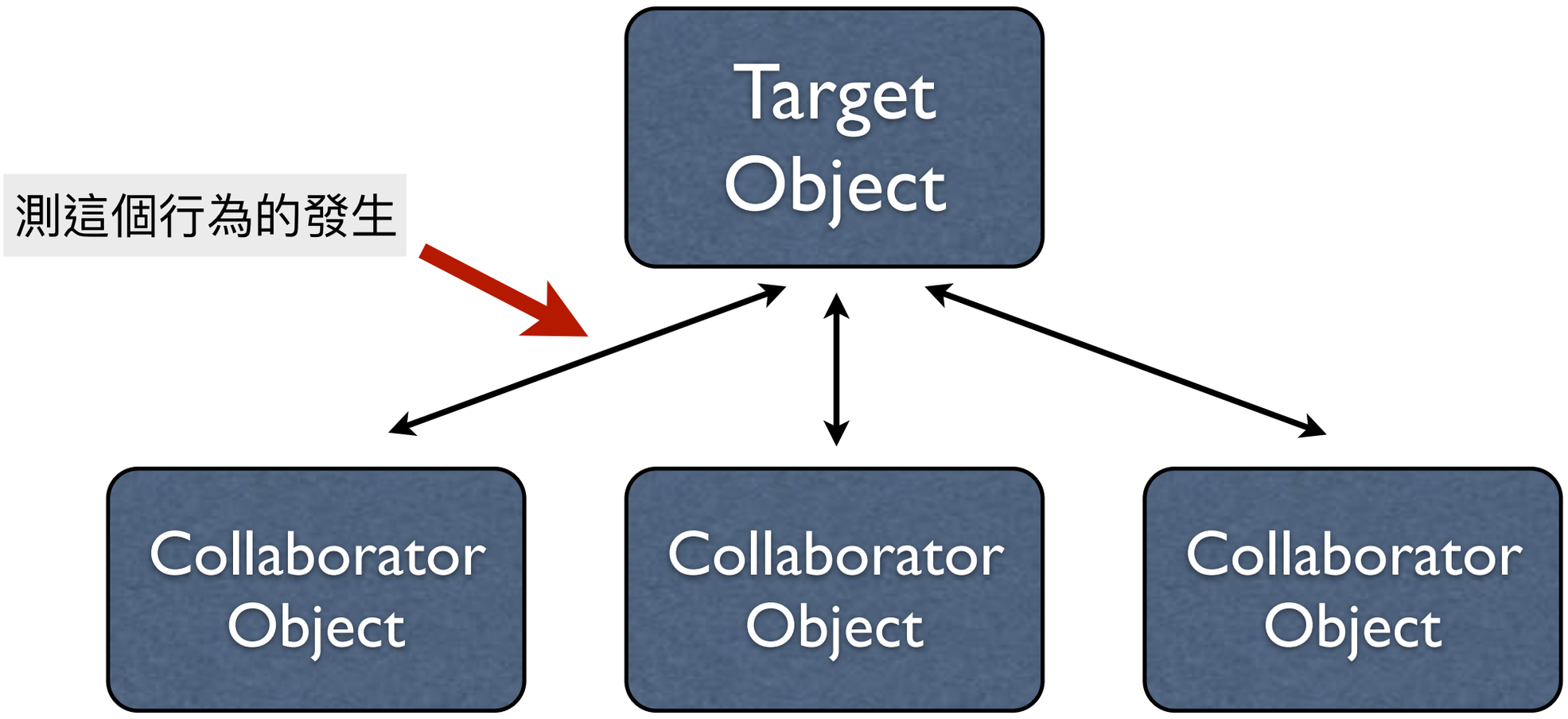
Given 給定條件
When 當事情發生
Then 則結果要是如何

Given 給定條件
Expect 預期會發生什麼
When 當事情發生

一般測試是檢查物件最後的狀態



Mocks 可以讓我們測試物件之間的行為



傳統 Classical Testing v.s. 完全 Mockist Testing

- Classical: 先決定你要完成的範圍，然後實作這個範圍的所有物件。
- Mockist: 只針對目標類別做測試及實作，不相干的用假物件。

Mockist 的缺點

- 你可能需要 stub 很多東西，nested stub 很醜
- stub 的回傳值可能難以建構
- 與 Collaborator 的行為太耦合，改介面就要跟著改一堆測試。
- 承上，即使忘了跟著改實作，單元測試還是會過... 例如在動態語言中，可能 Collaborator 後來改了方法名稱，但是這邊的測試仍然會過...

Mocks 如何用得好?

- 不要用 Mocks 來隔離內部物件，用在重要的邊界上，例如第三方服務
  - https://8thlight.com/blog/uncle-bob/2014/05/10/WhenToMock.html
- 最好不要直接 Mocks 第三方套件的介面，會增加相依性讓測試脆弱
  - 多包一層自己設計的介面
    - http://blog.carbonfive.com/2011/02/11/better-mocking-in-ruby/
- 有更高層級的整合測試，使用真的物件來做測試
- 採用 TDD `讓 API 設計可以趨向:
  - 不需要 stub 太多層
  - 回傳值簡單
  - 類別之間的耦合降低
  - 遵守 Law of Demeter

造假舉例：Logger

已知有一個 Logger 其 log 方法運作正常

```
# 測狀態
logger.log("Report generated")
expect(File.read("log.log")).to eq "Report generated"
# 如果 Logger 換別種實作就死了
```

```
# 測行為
expect(logger).to receive(:log).with(/Report generated/)
logger.log("Report generated")
```

造假舉例：
MVC 中的Controller 測試

```ruby
# 測狀態
it "should be created successful" do
  post :create, :name => "ihower"
  expect(response).to be_success
  expect(User.last.name).to eq("ihower") # 會真的存進資料庫
end
```

```ruby
# 測行為: 寫法很囉唆
it "should be created successful" do
  expect(Order).to receive(:create).with(:name => "ihower").and_return(true)
  post :create, :name => "ihower"
  expect(response).to be_success
end
```

Spies

- Mocks 將 expectation 的部分寫在開頭，跟一般的測試順序 Four-Phase Test 不一樣，有些人覺得很不直覺
- 可用 Spies 的 have_received 寫法，就會是你熟悉的順序
- https://relishapp.com/rspec/rspec-mocks/v/3-5/docs/basics/spies

用在 test double 上

```
RSpec.describe "have_received" do
  it "passes when the message has been received" do
    invitation = spy('invitation')
    invitation.deliver
    expect(invitation).to have_received(:deliver)
  end
end
```

也可用在 partial double 上

```
RSpec.describe "have_received" do
  it "passes when the expectation is met" do
    allow(Invitation).to receive(:deliver)
    Invitation.deliver
    expect(Invitation).to have_received(:deliver)
  end
end
```

Code Kata

- Train Reservation HTTP Client library
  - GET /train/{:id} 拿列車座位資料
  - POST /train/{:id}/reservation 定位
- Ticket Office Web Service (Part 2)

What have we learned?

- 利用 Mocks 處理測試邊界(第三方服務)
- 設計 Web Service API (Part 2 會繼續沿用)

補充: HTTP Client 測試

- 更一般性的 Mock Library https://github.com/bblimke/webmock
- 用錄的 https://github.com/vcr/vcr (for Facebook, Twitter, Google 已經存在的服務)

http://robots.thoughtbot.com/how-to-stub-external-services-in-tests
https://thoughtbot.com/upcase/videos/testing-interaction-with-3rd-party-apis

Reference:

- The RSpec Book
- The Rails 3 Way
- Foundation Rails 2
- xUnit Test Patterns
- everyday Rails Testing with RSpec
- http://pure-rspec-rubynation.heroku.com/
- http://jamesmead.org/talks/2007-07-09-introduction-to-mock-objects-in-ruby-at-lrug/
- http://martinfowler.com/articles/mocksArentStubs.html
- http://blog.rubybestpractices.com/posts/gregory/034-issue-5-testing-antipatterns.html
- http://blog.carbonfive.com/2011/02/11/better-mocking-in-ruby/

Thanks.