

投影片 <https://ihower.tw/presentation/ihower-jasmine.pdf>

Jasmine & TDD Tutorial

<https://ihower.tw>
2015/4

About me

- 張文鈿 a.k.a. ihowr
 - <http://ihower.tw>
 - <http://twitter.com/ihower>
- Instructor at ALPHA Camp
 - <https://alphacamp.co>
- Ruby on Rails Developer since 2006



Agenda

- Jasmine and TDD
 - 什麼是自動化測試
 - Jasmine 基本語法
 - TDD and Code Kata
 - Spies (Mocks and Stubs)
 - Async

什麼是測試？



Search



Home

Profile

Messages

Who To Follow



ihower ▾



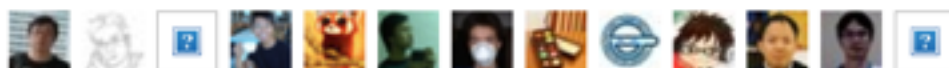
@ihower

Wen-Tien Chang

為什麼測試喜歡搞這麼多名目：單元測試、
驗收測試、需求測試、煙霧測試、回歸測
試、整合測試、系統測試、功能測試、確認
(validation)測試、白箱測試、黑箱測試、
灰箱測試、錯誤處理測試、壓力測試、效能
測試、安全性測試、相容性測試、使用性測
試、完整性測試、結構測試、安裝測試、

27 Sep via web ☆ Favorite ↻ Reply 🗑 Delete

Retweeted by sandzhang and 15 others



Sorry, I'm not QA guy :/

測試 Overview

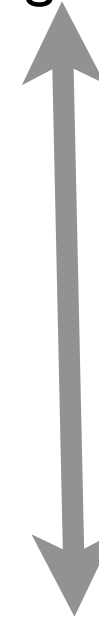
單元測試
Unit Test

整合測試
Integration Test

驗收測試
Acceptance Test

Verification

確認程式的執行結果有沒有對?
Are we writing the code right?



Validation

檢查這軟體有滿足用戶需求嗎?
Are we writing the right software?

單元測試
Unit Test

整合測試
Integration Test

驗收測試
Acceptance Test

測試個別的類別和函式結果正確

測試多個類別之間的互動正確

從用戶觀點測試整個軟體



單元測試
Unit Test

整合測試
Integration Test

驗收測試
Acceptance Test

白箱測試



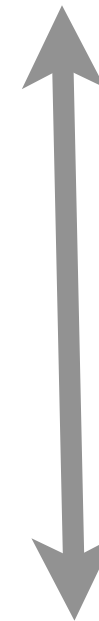
黑箱測試

單元測試
Unit Test

整合測試
Integration Test

驗收測試
Acceptance Test

Developer 開發者



QA 測試人員

Verification

確認程式的執行結果有沒有對?

Are we writing the code right?

我們天天都在檢查程式的結果有沒有對

- 程式寫完後，手動執行看看結果對不對、打開瀏覽器看看結果對不對？(如果你是web developer 的話)
- 聽說有人只檢查 `syntax` 語法沒錯就 `commit` 程式的？

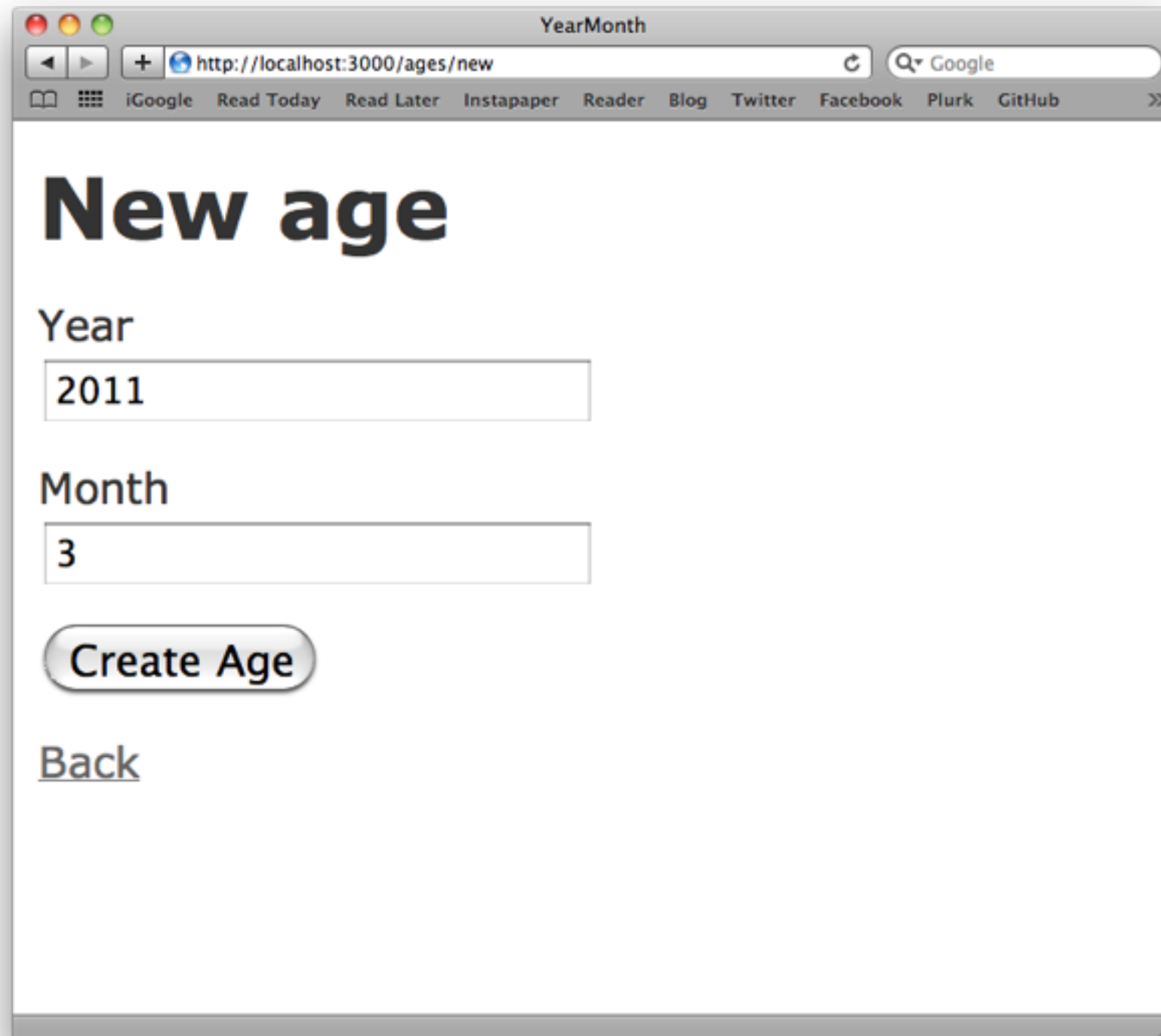
手動測試很沒效率

特別是複雜或 dependency 較多的程式，你可能會偷懶沒測試所有的執行路徑

有三種路徑，你得執行(手動測試)三次才能完整檢查

```
exports.year_month = function(year, month) {  
  
  if ( month > 12 ) {  
    if ( month % 12 == 0 ) {  
      year += (month - 12) / 12  
      month = 12  
    } else {  
      year += Math.floor( month / 12 )  
      month = month % 12  
    }  
  }  
  
  return [year, month]  
}
```

用介面手動測試?



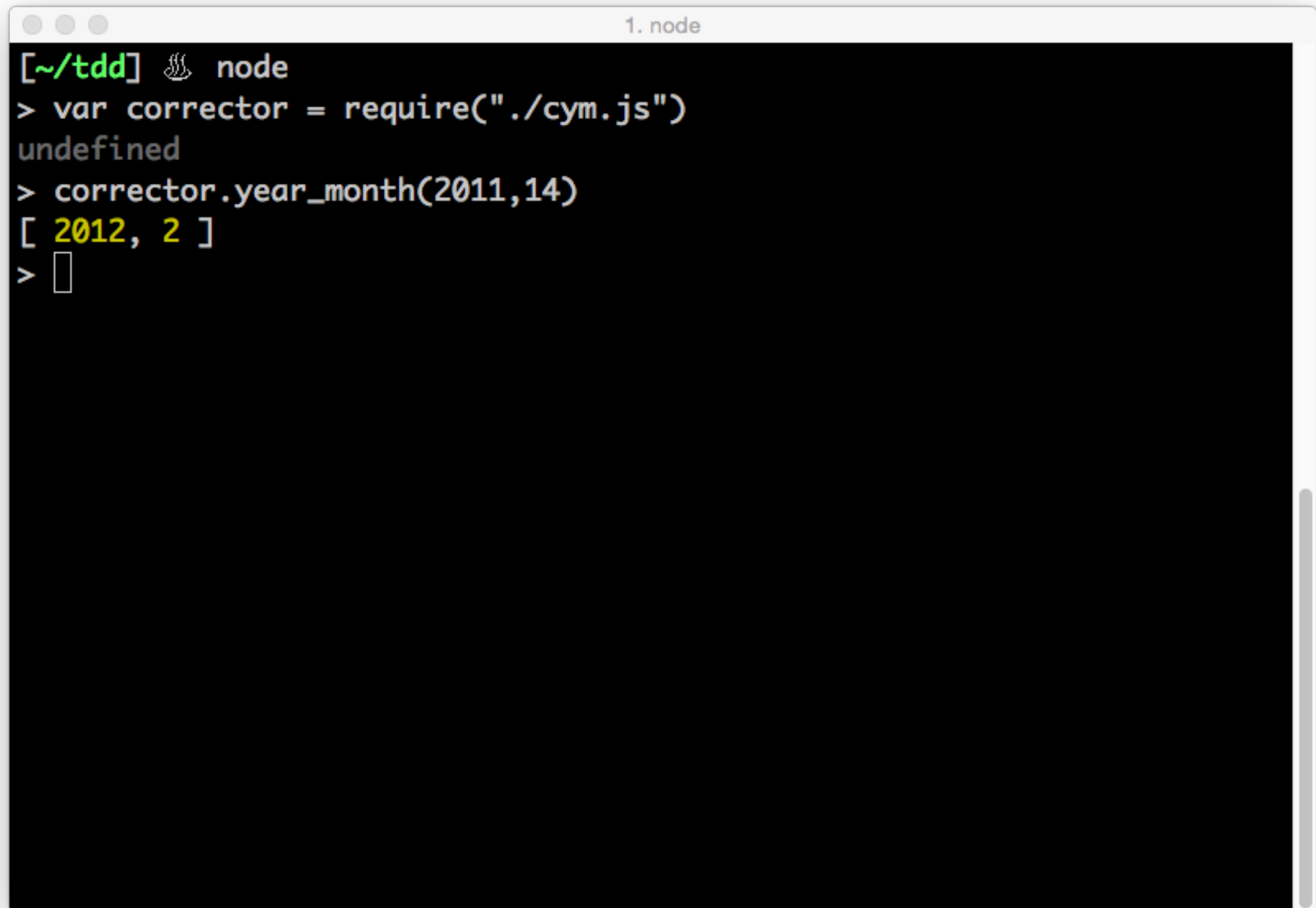
A screenshot of a web browser window titled "YearMonth". The address bar shows "http://localhost:3000/ages/new". The browser's toolbar includes a search bar with "Google" and a list of bookmarks: "iGoogle", "Read Today", "Read Later", "Instapaper", "Reader", "Blog", "Twitter", "Facebook", "Plurk", and "GitHub".

The main content area displays the heading "New age" in a large, bold font. Below this, there are two input fields:

- A "Year" label followed by an input field containing the text "2011".
- A "Month" label followed by an input field containing the text "3".

Below the input fields is a rounded button labeled "Create Age". At the bottom left of the form area is a link labeled "Back".

用 Console 介面手動測?



```
1. node
[~/tdd] node
> var corrector = require("./cym.js")
undefined
> corrector.year_month(2011,14)
[ 2012, 2 ]
> 
```

自動化測試

讓程式去測試程式

I. Instant Feedback

寫好測試後，很快就可以知道程式有沒有寫對

```
it("should should return correct year and month", function () {  
  expect(corrector.year_month(2011, 3)).toEqual( [2011, 3] );  
  expect(corrector.year_month(2011, 14)).toEqual( [2012, 2] );  
  expect(corrector.year_month(2011, 24)).toEqual( [2012, 12] );  
})
```

寫測試的時間
小於
debug 除錯的時間

2. 回歸測試及重構

- 隨著程式越寫越多，新加的程式或修改，會不會搞爛現有的功能？
- 重構的時候，會不會造成破壞？
- 如果有之前寫好的測試程式，那們就可以幫助我們檢查。

3. 幫助你設計 API

- 寫測試的最佳時機點：先寫測試再實作
- TDD (Test-driven development)
 - Red-Green-Refactor 流程
- 從呼叫者的角度去看待程式，關注介面，協助設計出好用的 API。

4. 一種程式文件

- 不知道程式的 API 怎麼呼叫使用?
- 查測試程式怎麼寫的，就知道了。

小結

- 你的程式不 Trivial -> 寫測試節省開發時間
- 你的程式不是用過即丟 -> 回歸測試和重構
- TDD -> 設計出可測試，更好的 API 介面
- API 怎麼用 -> 測試也是一種文件

怎麼寫測試?

xUnit Framework

xUnit Framework

每個程式語言都有這樣的框架

- test case 依序執行各個 test :
 - Setup 建立初始狀態
 - Exercise 執行要測的程式
 - Verify (assertion) 驗證狀態如預期
 - Teardown 結束清理資料

JavaScript Unit Test Framework

- Jasmine
- Mocha
- QUnit

Jasmine

<http://jasmine.github.io/>

- JavaScript 單元測試框架，強調 BDD
 - Semantic Code：容易描述測試目的
 - Self-documenting：可執行的規格文件
- 前端、後端 JavaScript 都支援

Jasmine 寫法

```
describe("order", function () {
```

```
    var order = new Order();
```

```
    describe("when initialized", function() {
```

```
        it("should have default status is New", function () {  
→      expect(order.status).toBe("New");  
        });
```

```
        it("should have default amount is 0", function () {  
→      expect(order.amount).toBe(0);  
        });
```

```
    });  
});
```

Learn Jasmine

- syntax
- syntax
- syntax
- more syntax

describe 幫助你組織分類

要測的東西是什麼?

```
describe("A Order", function(){  
    //...  
})
```


可以 Nested 多層表示不同情境

```
describe("A Order", function(){  
    describe("#amount", function(){  
        //...  
    })  
})
```

每個 it 就是一小段測試

Assertions 又叫作 Expectations

加入 it

```
describe("order", function () {
```

```
  describe("#amount", function() {
```

```
    describe("when user is vip",function(){
```

→

```
      it("should discount five percent if total > 1000", function () {  
        //...  
      })
```

→

```
      it("should discount ten percent if total > 10000", function () {  
        //...  
      })  
    })
```

```
  describe("when user is not vip", function() {
```

→

```
    it("should discount three percent if total > 10000", function () {  
      //...  
    })  
  })  
})
```

```
})
```

expect(...) 或

expect.not

搭配 Matcher 定義你的期望

搭配不同 Matcher

- toBe
- toEqual
- toMatch
- toBeDefined
- toBeUndefined
- toBeTruthy
- toBeFalsy
- toContain
- toThrow

```
describe("order", function () {  
    describe("#amount", function() {  
        describe("when user is vip",function(){  
            it("should discount five percent if total >= 1000", function () {  
                var user = new User( { is_vip: true } )  
                var order = new Order( user, { total: 2000 } )  
                → expect(order.amount).toBe(1900)  
            })  
        })  
    })  
})
```

輸出結果(red)

```
1. bash
[~/kata/js/order] ❏ jasmine-node spec/
..FFF

Failures:

1) order #amount when user is vip should discount five percent if total >= 1000
  Message:
    Expected 0 to be 1900.
  Stacktrace:
    Error: Expected 0 to be 1900.
    at null.<anonymous> (/Users/ihower/kata/js/order/spec/order_spec.js:25:30)

2) order #amount when user is vip should discount ten percent if total >= 10000
  Message:
    Expected 0 to be 9000.
  Stacktrace:
    Error: Expected 0 to be 9000.
    at null.<anonymous> (/Users/ihower/kata/js/order/spec/order_spec.js:31:30)

3) order #amount when user is not vip should discount three percent if total >= 10000
  Message:
    Expected 0 to be 9700.
  Stacktrace:
    Error: Expected 0 to be 9700.
    at null.<anonymous> (/Users/ihower/kata/js/order/spec/order_spec.js:39:30)

Finished in 0.024 seconds
5 tests, 5 assertions, 3 failures, 0 skipped

[~/kata/js/order] ❏
```

(狀態顯示為在寫 Order 主程式)

輸出結果(green)

```
1. bash
[~/kata/js/order] (✓ finish) 🍳 jasmine-node spec/ --verbose

order - 7 ms

  when initialized - 6 ms
    should have default status is New - 4 ms
    should have default amount is 0 - 2 ms

  #amount - 1 ms

    when user is vip - 1 ms
      should discount five percent if total >= 1000 - 1 ms
      should discount ten percent if total >= 10000 - 0 ms

    when user is not vip - 0 ms
      should discount three percent if total >= 10000 - 0 ms

Finished in 0.015 seconds
5 tests, 5 assertions, 0 failures, 0 skipped

[~/kata/js/order] (✓ finish) 🍳 □
```

漂亮的程式文件

before 和 after

- 如同 xUnit 框架的 setup 和 teardown
- beforeEach 每段 it 之前執行
- afterEach 每段 it 之後執行

```
describe("order", function () {
```

```
  describe("#amount", function() {  
    var user, order;
```

```
    describe("when user is vip",function(){
```

➔

```
      beforeEach(function(){  
        user = new User( { is_vip: true } )  
      })
```

```
      it("should discount five percent if total >= 1000", function () {  
        order = new Order( user, { total: 2000 } )  
        expect(order.amount()).toBe(1900)  
      })
```

```
      it("should discount ten percent if total >= 10000", function () {  
        var user = new User( { is_vip: true } )  
        var order = new Order( user, { total: 10000 } )  
        expect(order.amount()).toBe(9000)  
      })
```

```
    })
```

```
  })
```

```
}
```

pending

可以先列出來打算要寫的測試

```
describe("order", function () {
```

```
→ xdescribe("#paid", function(){  
    it("should be false if status is new")
```

```
→ xit("should be true if status is paid or shipping", function(){  
    //...  
})  
})  
})
```

```
[~/ruby] $ rspec order_spec.rb -fs
```

Order

#amount

when user is vip

should discount five percent if total >= 1000

should discount ten percent if total >= 10000

when user is not vip

should discount three percent if total > 10000

#paid?

should be false if status is new (PENDING: Not Yet Implemented)

should be true if status is paid or shipping (PENDING: No reason given)

Pending:

Order#paid? should be false if status is new

Not Yet Implemented

./order_spec.rb:39

Order#paid? should be true if status is paid or shipping

No reason given

./order_spec.rb:41

Finished in 0.00137 seconds

5 examples, 0 failures, 2 pending

```
[~/ruby] $
```

語法複習

- describe
- it
- expect(實際).toBe(預期)
- expect(實際).toEqual(預期)

Code Kata

- 一種練習方法，透過小型程式題目進行鍛鍊，就像學功夫套拳，重複練功一樣
- 除了自己練，也可以 pair-programming
- TDD 需要練習才能抓到訣竅，與其說它是一種測試方式，不如說它更像一種設計手法
- 抓到 TDD 測試精神和訣竅，比學再多漂亮的語法糖更重要

準備 Kata 環境

<https://github.com/ihower/js-kata>

- `npm install jasmine -g`
 - `mkdir project_name; cd project_name;`
 - `jasmine init`
 - `jasmine`
- 或 `npm install jasmine-node -g`
 - `jasmine-node spec/ --autotest --verbose --color --watch ./`
 - **Continous Testing:** 程式一修改完存檔，自動跑對應的測試：節省時間，立即回饋
 - `jasmine-node` 用的是舊版的 `jasmine 1.3`

TDD 訣竅一：

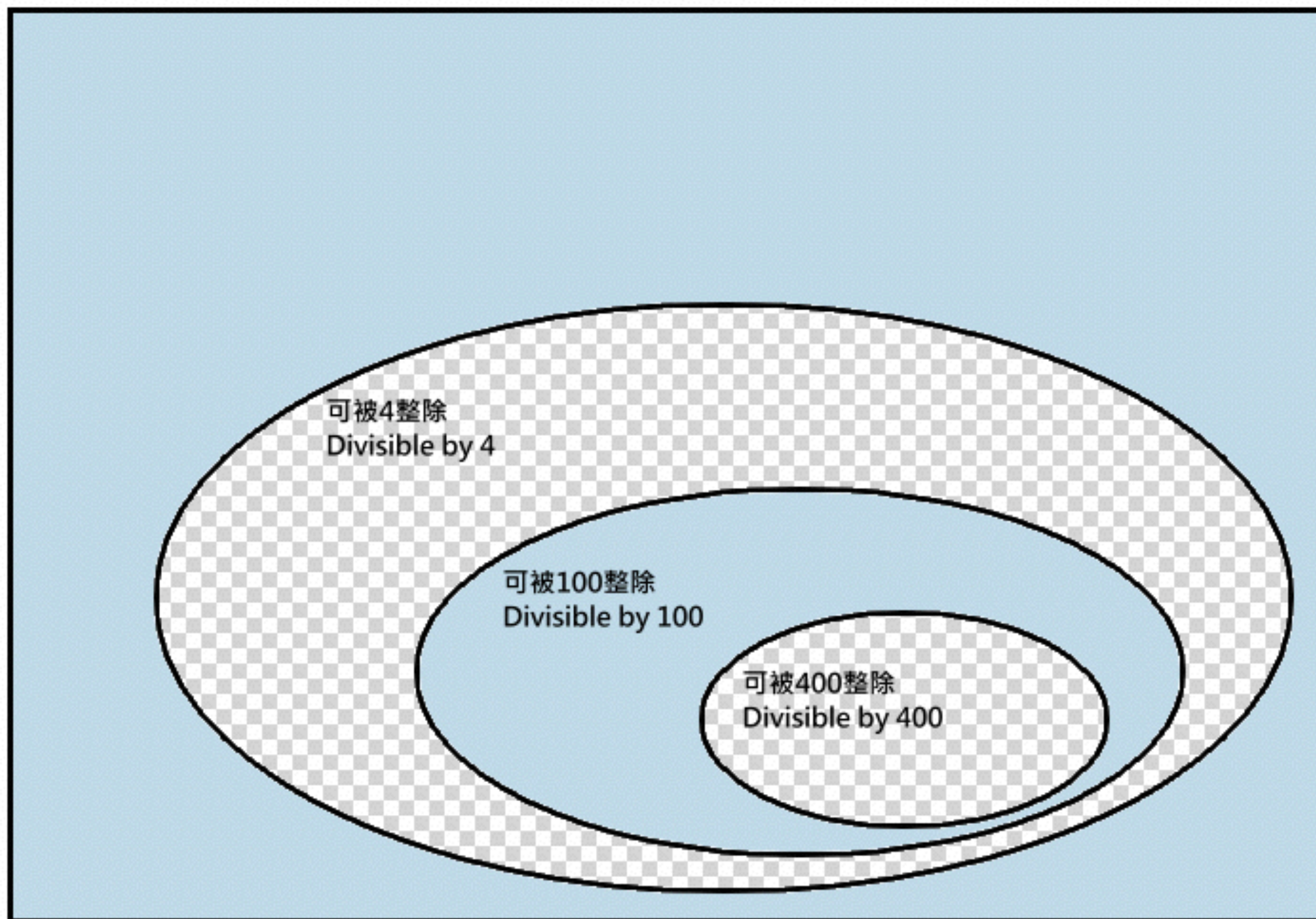
Red-Green-Refactor
development cycle

FizzBuzz

- 逢三整除，輸出 Fizz
- 逢五整除，輸出 Buzz
- 逢三五整除，輸出 FizzBuzz

Leap Years

- 判斷閏年
 - 西元年份除以400可整除，為閏年。
 - 西元年份除以4可整除但除以100不可整除，為閏年。
 - 其他則是平年



平年 Common year 閏年 Leap year

Roman Numerals

- 轉換羅馬數字
 - 1 -> I
 - 4 -> IV
 - 5 -> V
 - 9 -> IX
 - 10 -> X
 - 20 -> XX

What have we learned?

- 一個 it 裡面只有一種測試目的，最好就只有一個 expectation
- 思考哪些是關鍵有用的測試案例
- 要先從測試 failed 失敗案例開始
 - 確保每個測試都有效益，不會發生砍掉實作卻沒有造成任何測試失敗
- 一開始的實作不一定要先直攻一般解，可以一步一步在 cycle 中進行思考和重構
- 測試程式碼的可讀性比 DRY 更重要
- 安全重構：無論是改實作或是改測試當時的狀態應該要維持 Green

As the tests get more specific, the code gets more generic.

Programmers make specific cases work by writing code that makes the general case work.

Three Laws of TDD

- 一定是先寫一個不通過的單元測試，才開始實作功能
- 每次只新加一個單元測試，只需要剛剛好不通過即可，不要一次加多個測試情境
- 每次實作功能時，只需要剛剛好通過測試即可，不多也不少

TDD 訣竅二：

讓測試引導 API 設計

Bowling Game

<http://www.sportcalculators.com/bowling-score-calculator>

- 計算保齡球分數
 - X (strike) 累加後兩球分數
 - / (spare) 累加後一球分數

BowlingGame

```
# roll
# roll_many(1,2,3,4)
# or roll_many("1234512345")
# finished?
# score
```

What have we learned?

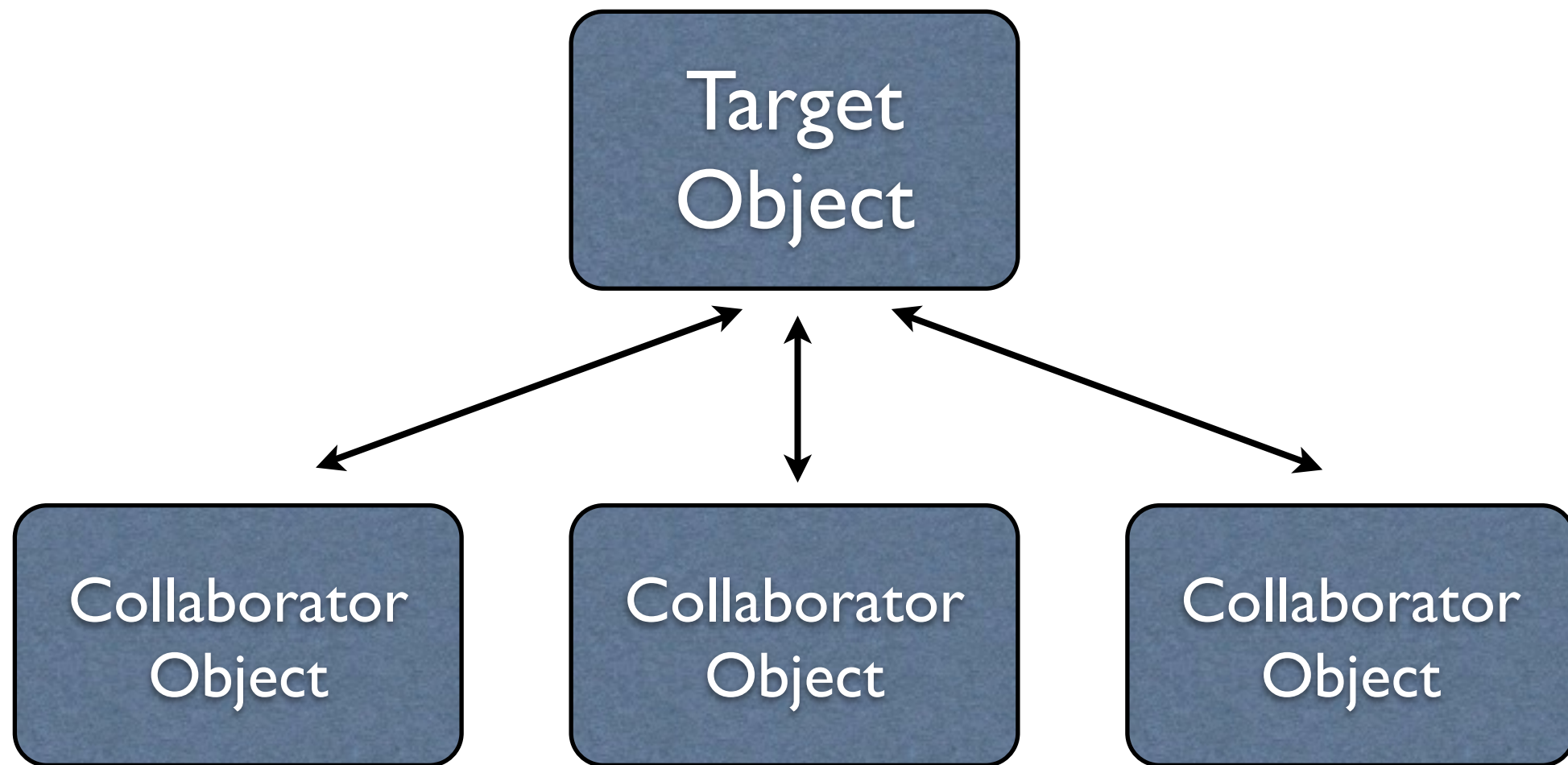
- 透過寫測試，定義出哪些是類別公開的介面(API)
- 測試碼方便呼叫的API，就是好API
- 不需要公開的，定義成 `private` 方法，讓實作有更好的物件導向封裝。
- 不需要針對 `private` 方法直接寫單元測試，而是透過 `public` 方法間接測試

Stub and Mock

用假的物件替換真正的物件，作為測試之用

物件導向是

物件和物件之間的互動



當你在寫目標類別的測試和實作時，這些 Collaborator...

- 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~~

使用假物件和假函式

- 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。

Jasmine 的 SpyOn

```
var foo = {  
  bar: function() { console.log("real") },  
};
```

➔ `spyOn(foo, 'bar').and.returnValue(456);`

// 或

➔ `spyOn(foo, 'bar').and.callFake(function(){ return 456; });`

測試狀態 Stub

```
describe("#receiver_name", function(){  
  it("should be user name", function(){  
    user = new User();  
    order = new Order(user);
```



```
    spyOn(user, "full_name").and.returnValue("GG")  
  
    expect( order.receiver_name() ).toBe("GG")  
  })  
})
```


測試行為 Mock

如果沒被呼叫到，就算測試失敗

```
describe("#ship!", function(){
  it("should call ezship API", function(){
    user = new User();
    ezcat = { deliver: function(user) { ... } };
    order = new Order(user, { gateway: ezcat } );

    spyOn(ezcat, "deliver")

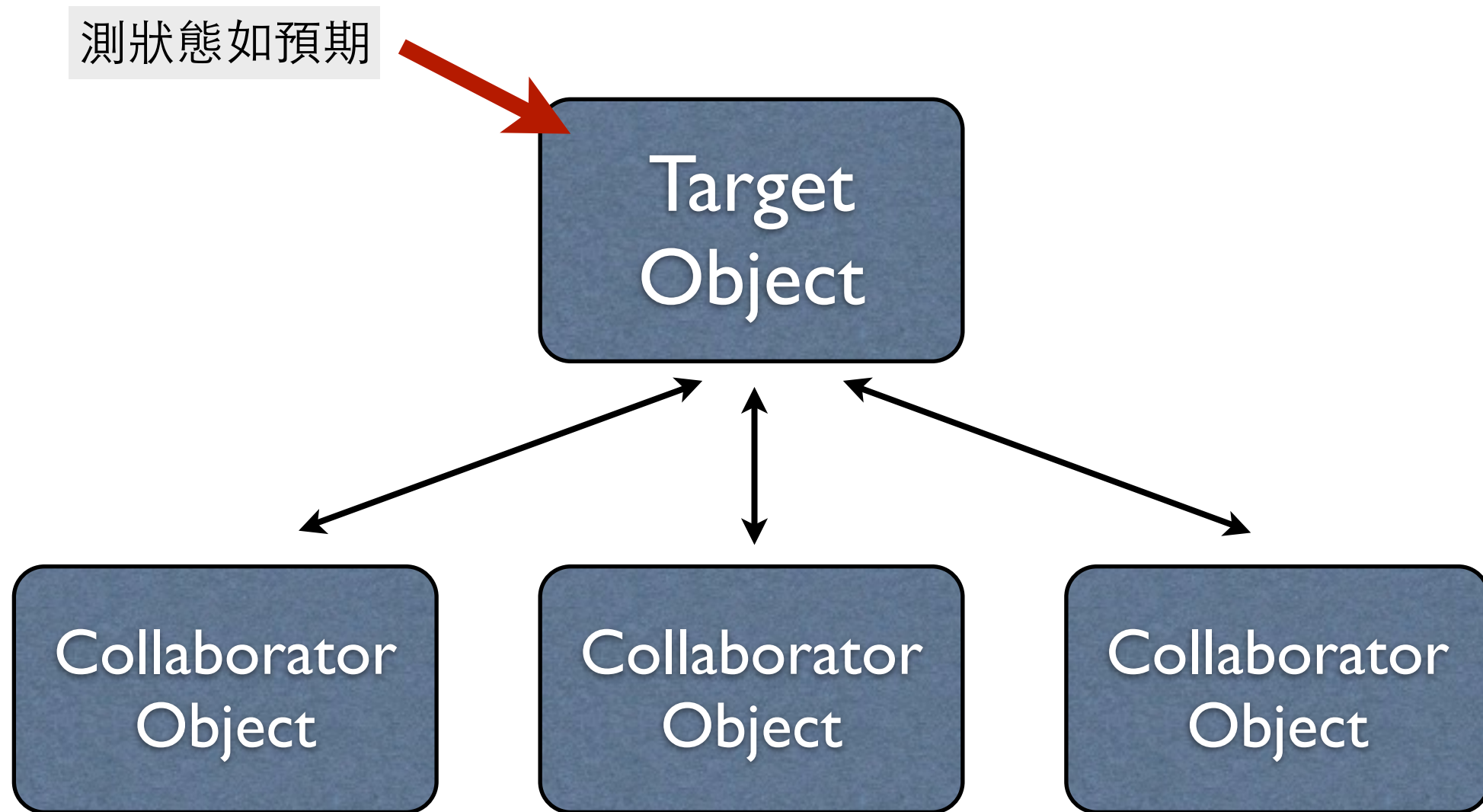
    order.ship();
```

```
→ expect( ezcat.deliver ).toHaveBeenCalledWith(user);
    })
  })
```

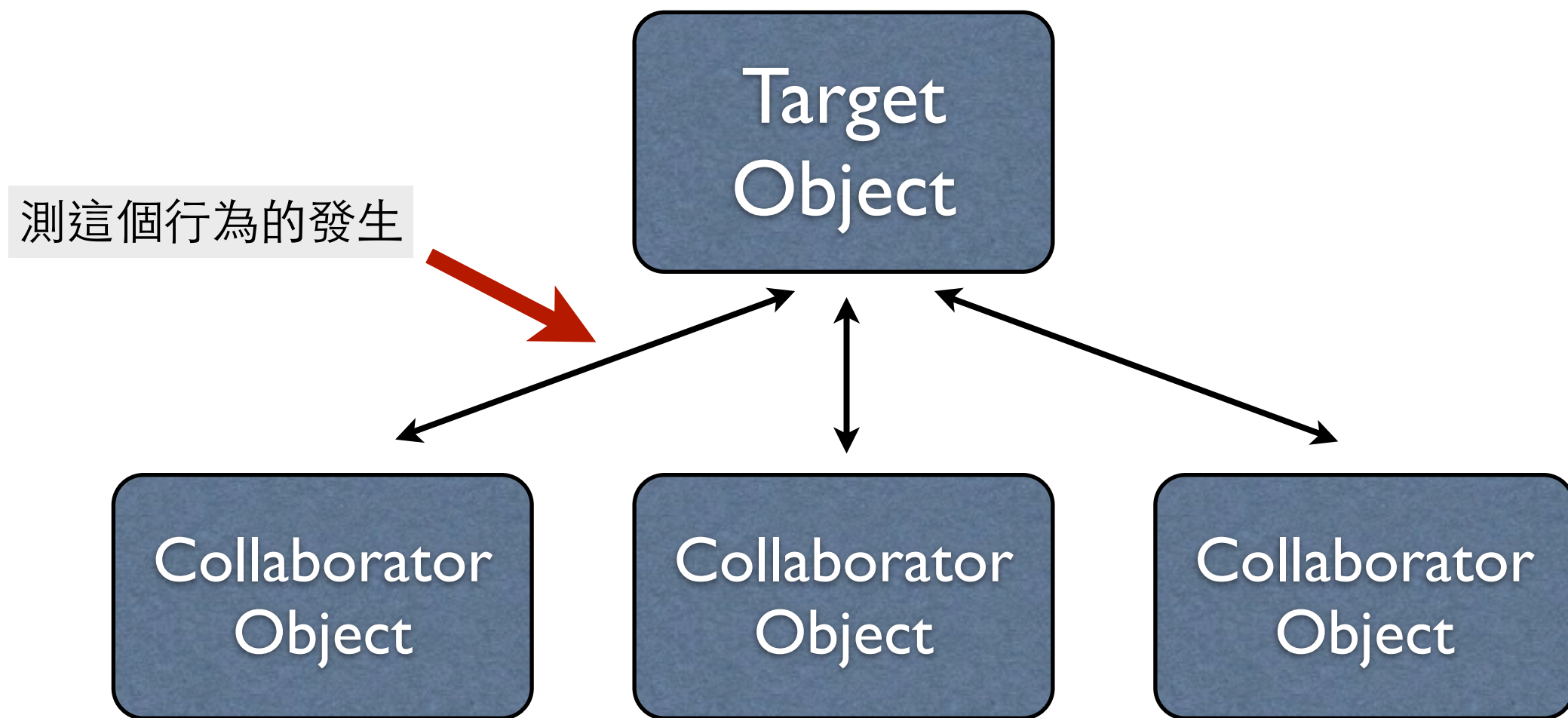
用來測試行為的發生

```
Order.prototype.ship = function(){  
    this.gateway.deliver(this.user);  
}
```

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



Classical Testing v.s. 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，不相干的用假回傳。

Mockist 缺點

- 你可能需要 stub 很多東西，nested stub 非常不好
- 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。

如何趨吉避凶？

- 必須擁有更高層級的整合測試，使用真的物件來做測試。
- 採用 TDD 思維讓 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

小心!

- 不要用 Mocks 來完全隔離內部 Internal dependencies
- 不要用 Mocks 來取代還沒實作的物件

有限度使用

- 造真物件不難的時候，造假的時間拿去造真的。
有需要再用 Stubbing/Mocking。
- 只在 Collaborator 不好控制的時候，Stub 它
- 只在 Collaborator 還沒實作的時候，Mock 它
- Collaborator 的狀態不好檢查或它的實作可能會改變，這時候才改成測試行為。

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(fs.readFileSync("log.log")).toBe("Report generated")`

如果 Logger 換別種實作就死了

測行為

```
logger.log("Report generated")
```

→ `expect(logger.log).toHaveBeenCalledWith("Report generated")`

Code Kata

- Train Reservation HTTP Client library
 - GET /train/{:id} 拿列車座位資料
 - POST /train/{:id}/reservation 定位

ProTip: 如何測試 Async Code?

```
exports.async_generate_awesome = function(cb){  
  
  setTimeout( function(){  
    cb("awesome");  
  }, 1000)  
  
}
```

done 方法

```
var kata = require("../kata")

describe("async_generate_awesome", function () {

  it("should async generate string", function (done) {
    kata.async_generate_awesome(function(str){
      expect(str).toBe("awesome");
      done();
    });
  });
});
```

What have we learned?

- 利用 Done() 處理非同步函式
- 利用 Spies Stub 處理測試邊界(第三方服務)

Reference:

- <http://jasmine.github.io/>
- The RSpec Book
- The Rails 3 Way
- Foundation Rails 2
- xUnit Test Patterns
- everyday Rails Testing with RSpec
- <http://pure-rspec-rubynation.herokuapp.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.

<https://github.com/ihowser/js-kata> 的 finish branch 有完成的範例程式