


 ĐAI HỌC BÁCH KHOA HÀ NỘI  
 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG


# IT4490 – Thiết kế và xây dựng phần mềm

## Bài 8. Kiểm thử đơn vị

1

## Nội dung


1. Tổng quan về kiểm thử
2. Kiểm thử đơn vị
3. Kiểm thử tích hợp


 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

2

### Kiểm thử (Testing)

- ❖ “[T]he means by which the presence, quality, or genuineness of anything is determined; a means of trial.” –[dictionary.com](https://www.dictionary.com)
- ❖ **Kiểm thử phần mềm** thực thi một chương trình để xác định xem một thuộc tính của chương trình có đạt hay không
- ❖ Một test **vượt qua / passes** [hoặc **thất bại / fails**] nếu thuộc tính **đạt được / holds** [hoặc **không đạt được / doesn't hold**] trong lần chạy đó


 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG


3

### Software Quality Assurance (QA)

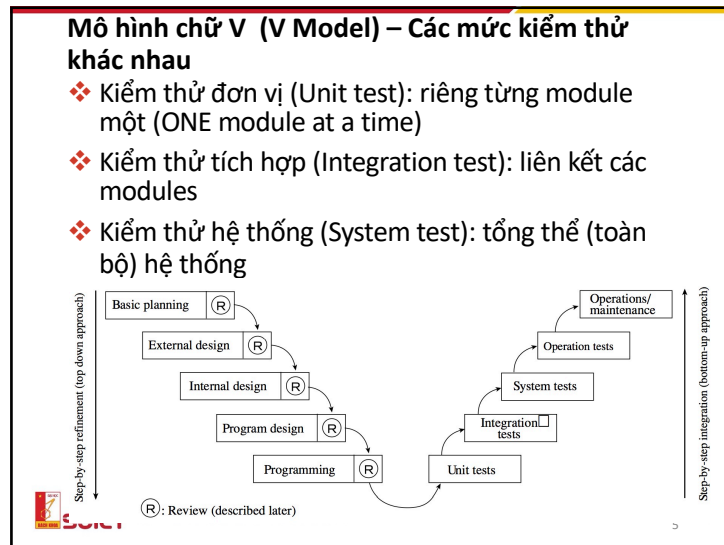
#### Kiểm thử kết hợp với các hoạt động khác bao gồm

- ❖ Phân tích tĩnh (đánh giá mã nguồn mà không cần phải thực thi chúng)
- ❖ Chứng minh tính đúng đắn (các định lý về những thuộc tính của chương trình)
- ❖ Xét duyệt mã nguồn (mỗi người xét duyệt mã nguồn của những người khác)
- ❖ Quy trình phần mềm (đặt cấu trúc vào vòng đời phát triển)
- ❖ ... và nhiều cách khác để tìm ra vấn đề và tăng cường sự tự tin

**No single activity or approach can guarantee software quality**


 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

4



5

**Các mức kiểm thử (Test levels)**

- ❖ **Kiểm thử đơn vị (Unit Testing):** Mỗi đơn vị (lớp, phương thức, v.v.) có làm những gì nó phải làm không?
  - Đơn vị lập trình nhỏ nhất
  - Chiến lược: Kiểm thử hộp đen và kiểm thử hộp trắng
  - Các kỹ thuật, các công cụ
- ❖ **Kiểm thử tích hợp (Integration Testing):** bạn có nhận được kết quả mong đợi khi các bộ phận được kết hợp với nhau?
  - Các chiến lược: Kiểm thử từ dưới lên (Bottom-up), kiểm thử từ trên xuống (top-down)
- ❖ **Kiểm thử hệ thống (System Testing):** hệ thống tổng thể có hoạt động không?
- ❖ **Kiểm thử chấp nhận (Acceptance Testing):** nó có phù hợp với các yêu cầu của người dùng?

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

6

**Nội dung**

1. Tổng quan về kiểm thử
- ➔ 2. Kiểm thử đơn vị
3. Kiểm thử tích hợp

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

7

**2.1. Các phương pháp tiếp cận kiểm thử đơn vị**  
**Kiểm thử hộp đen và hộp trắng**

- A. Chọn dữ liệu đầu vào ("test inputs")
- B. Định nghĩa đầu ra mong đợi ("soict")
- C. Thực thi đơn vị ("SUT" or "software under test") trên đầu vào và ghi nhận các kết quả
- D. Kiểm tra kết quả có trái với đầu ra mong đợi ("soict")

**Hộp đen (Black box)**  
Phải chọn đầu vào mà không có kiến thức về việc thực thi

**Hộp trắng (White box)**  
Có thể chọn đầu vào với kiến thức về việc thực thi

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

8

## Không chỉ black-and-white, mà...

**Black box**  
Must choose inputs *without knowledge* of the implementation

- ❖ Phải tập trung vào hành vi của SUT
- ❖ Cần một “soic”
  - Hoặc ít nhất là một kỳ vọng về việc liệu **một ngoại lệ** có được ném ra hay không

**White box**  
Can choose inputs *with knowledge* of the implementation

- ❖ Sử dụng phổ biến: bao phủ (**coverage**)
- ❖ Ý tưởng cơ bản: nếu bộ kiểm thử (test suite) của bạn khiến một câu lệnh không bao giờ được thực thi, thì câu lệnh đó có thể bị lỗi

 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG 9

9

## Các thuật ngữ

**Test Plan** = **Test Strategy** + **Test Logistics**


↓

Test Strategy is Approach  
i.e.  
What, Why, When  
& How  
in Testing

↓

Test Logistics –  
Who in Testing?

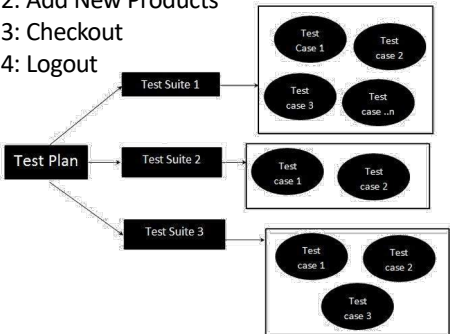
- ❖ Trường hợp kiểm thử (Test case)
  - một tập hợp các điều kiện / biến để xác định xem một hệ thống đang được kiểm thử có đáp ứng các yêu cầu hoặc hoạt động chính xác hay không
- ❖ Bộ kiểm thử (Test suite)
  - một tập hợp các trường hợp thử nghiệm liên quan đến cùng một công việc thử nghiệm
- ❖ Kế hoạch kiểm thử (Test plan)
  - tài liệu mô tả cách tiếp cận kiểm thử và phương pháp luận đang được sử dụng để thử nghiệm dự án, rủi ro, phạm vi kiểm thử, các công cụ cụ thể


 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG 10

10

## Bộ kiểm thử (Test suite)

- ❖ Ví dụ về test suite
  - Test case 1: Login
  - Test case 2: Add New Products
  - Test case 3: Checkout
  - Test case 4: Logout




 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG 11

11

## Các kỹ thuật kiểm thử đơn vị

- ❖ Dành cho thiết kế trường hợp thử
- ❖ (2.2) Các kỹ thuật kiểm thử cho kiểm thử hộp đen (Black Box Test)
  - Phân tích phân vùng tương đương (Equivalence Partitioning Analysis)
  - Phân tích giá trị biên (Boundary-value Analysis)
  - Bảng quyết định (Decision Table)
  - Kiểm thử dựa trên ca sử dụng (Use Case-based Test)
- ❖ (2.3) Các kỹ thuật kiểm thử cho kiểm thử hộp trắng (White Box Test)
  - Kiểm thử luồng điều khiển với phủ C0, C1 (Control Flow Test with C0, C1 coverage)
  - Kiểm thử phủ biểu đồ tuần tự (Sequence chart coverage test)

 VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG 12

12

## 2.2. Các kỹ thuật kiểm thử hộp đen

### 2.2.1. Phân vùng tương đương

- ❖ Tạo các trường hợp thử nghiệm bao trùm bằng cách phân tích không gian dữ liệu đầu vào và chia thành các lớp tương đương
  - Không gian điều kiện đầu vào được phân chia thành các lớp tương đương
  - Mọi đầu vào được lấy từ một lớp tương đương tạo ra cùng một kết quả

13

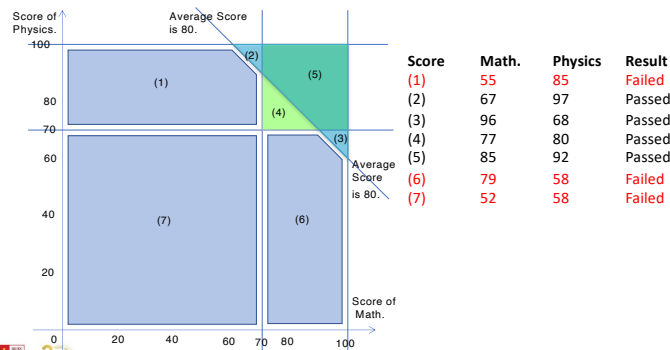
## Ví dụ: Chương trình Đánh giá Kỳ thi

- ❖ Tên chương trình: "Chương trình đánh giá kỳ thi"
- ❖ Môn học: Hai môn Toán và Vật lý
- ❖ Đặc tả:
  - Để kỳ thi nếu
    - điểm của cả toán và vật lý lớn hơn hoặc bằng 70 trên 100
  - hoặc,
  - điểm trung bình của toán học và vật lý lớn hơn hoặc bằng 80 trên 100
  - Trượt => nếu ngược lại

14

## Phân vùng tương đương của không gian đầu vào và các trường hợp kiểm thử

- ❖ 7 lớp tương đương => ít nhất 7 trường hợp kiểm thử / dữ liệu



15

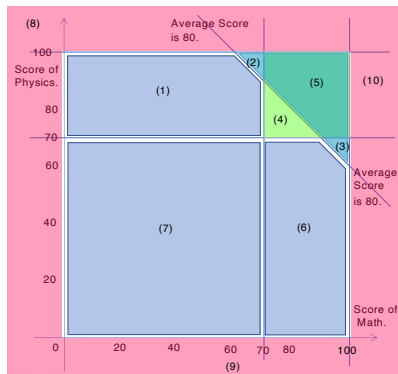
## Phân vùng tương đương Thảo luận và phân tích bổ sung

- ❖ Chúng ta có thành công không?
  - Không, chúng ta không thành công! Tại sao?
    - Còn thiếu một thứ!
- ❖ Phạm vi không gian đầu vào được phân tích là không đủ!
- ❖ Chúng ta phải thêm "Giá trị không hợp lệ" làm dữ liệu kiểm thử.
  - Ví dụ: một số mẫu "Giá trị không hợp lệ".
    - (8) Toán = -15, Vật lý = 120      Cả hai điểm đều không hợp lệ.
    - (9) Toán = 68, Vật lý = -66      Điểm Vật lý không hợp lệ.
    - (10) Toán = 118, Vật lý = 85      Điểm Toán không hợp lệ.

16

## Thêm các lớp tương đương

### ❖ Thêm 3 test cases/data



Một số dữ liệu không hợp lệ đã được thêm.

Score	Math.	Physics	Result
(1)	55	85	Failed
(2)	67	97	Passed
(3)	96	68	Passed
(4)	77	80	Passed
(5)	85	92	Passed
(6)	79	58	Failed
(7)	52	58	Failed
(8)	-15	120	Invalid
(9)	68	-66	Invalid
(10)	118	85	Invalid

17

## Phân tích và thảo luận

- ❖ Chúng ta đã cố gắng tạo các trường hợp thử nghiệm bao gồm dựa trên đặc điểm kỹ thuật bên ngoài.
  - Thành công? "Đúng"!

- ❖ Câu hỏi tiếp theo. Các trường hợp / dữ liệu thử nghiệm có đầy đủ hiệu quả?
  - Chúng ta phải tập trung vào nơi còn nhiều khiếm khuyết, phải không?
  - Chỗ đó là ở đâu?

→ "Phân tích giá trị biên"

18

## 2.2. Các kỹ thuật kiểm thử hộp đen

### 2.2.2. Phân tích giá trị biên

- ❖ Trích xuất dữ liệu kiểm tra được mong đợi bằng cách phân tích các giá trị đầu vào biên => Dữ liệu kiểm tra hiệu quả
  - Giá trị biên có thể phát hiện nhiều khiếm khuyết một cách hiệu quả

→ Ví dụ mathematics/physics score bằng 69 và 70

- Lập trình viên đã mô tả đoạn mã sai như sau:

```
if (mathscore > 70) {
    .....
}
```

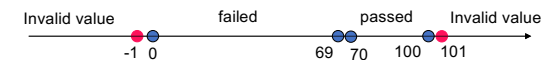
- Thay vì viết mã đúng như sau;

```
if (mathscore >= 70) {
    .....
}
```

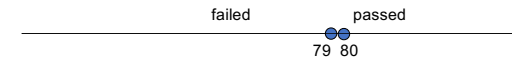
19

## Ví dụ: Phân tích giá trị biên

- ❖ Các giá trị biên của điểm toán trong case study:



- ❖ Còn về phân tích giá trị biên cho điểm trung bình của toán và vật lý?



20

## 2.2. Các kỹ thuật kiểm thử hộp đen

### 2.2.3. Bảng quyết định

- ❖ Mỗi quan hệ giữa các điều kiện và nội dung của quá trình xử lý được thể hiện dưới dạng một bảng
- ❖ Bảng quyết định là một công cụ dạng bảng được sử dụng khi các điều kiện phức tạp được kết hợp

21

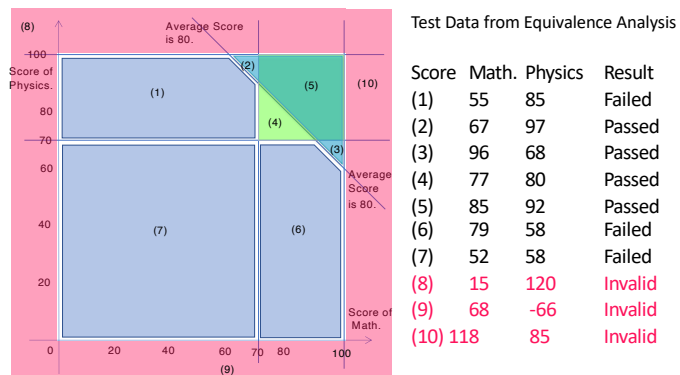
### Ví dụ: Bảng quyết định

- ❖ Các điều kiện để tạo báo cáo từ tệp nhân viên

Under age 30	Y	Y	N	N
Male	Y	N	Y	N
Married	N	Y	Y	N
Output Report 1	-	X	-	-
Output Report 2	-	-	-	X
Output Report 3	X	-	-	-
Output Report 4	-	-	X	-

22

### Bảng quyết định cho “Examination Judgement”???



23

### Bảng quyết định cho “Examination Judgement”

Condition1: Mathematics score>=>70

Condition2: Physics score=>70

Condition3: Average of Mathematics, and Physics =>80

	TC5	TC4	TC3	TC6	TC2	TC1	TCNG	TC7
Condition1	True	True	True	True	False	False	False	False
Condition2	True	True	False	False	True	True	False	False
Condition3	True	False	True	False	True	False	True(none)	False
"Passed"	Yes	Yes	Yes	---	Yes	---	N/A	--
"Failed"	---	---	---	Yes	---	Yes	N/A	Yes

24

## Bảng quyết định cho “Examination Judgement”

### ❖ Dữ liệu vào không hợp lệ (integer)

- Condition1: Mathematics score = valid that means “0<= the score <= 100”
- Condition2: Physics score = valid that means “0<= the score <= 100”

	TC1	TC2	TC3	TC4
Condition1	Valid Invalid	Valid	Invalid	
Condition2	Valid Valid	Invalid	Invalid	
“Normal results”	Yes	---	---	---
“Error message math”	---	Yes	---	Yes
“Error message phys”	---	---	Yes	Yes

Nếu cả điểm toán và điểm vật lý đều không hợp lệ, hai thông báo sẽ được xuất ra. Đây có phải là một đặc tả chính xác không? Hãy xác nhận nó?

25

## 2.2. Các kỹ thuật kiểm thử hộp đen

### 2.2.4. Kiểm thử cho Use case

#### ❖ ???

#### ❖ Ví dụ. Bảng quyết định cho Login

- Các điều kiện
  - ???
- Các kết quả
  - ???

#### ❖ Ví dụ. Phân tích giá trị biên

- ?

26

## Các trường hợp kiểm thử cho “Log in”

### ❖ “Thành công”

- Mã PIN đúng

### ❖ “Thất bại”

- Mã PIN sai và số lần sai < 3

### ❖ “Khoá tài khoản”

- Mã PIN sai và số lần sai = 3

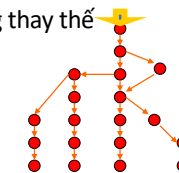
Mã PIN đúng	Y	Y	N	N
Số lần sai < 3	Y	N	Y	N
“Thành công”	x	N/A	-	-
“Thất bại”	-	N/A	x	-
“Khoá tài khoản”	-	N/A	-	x

- Phân tích vùng biên? Số lần sai = 2, 4 (?)

27

## Tạo các trường hợp kiểm thử từ các ca sử dụng

- Xác định tất cả các kịch bản cho trường hợp sử dụng nhất định
- Các kịch bản thay thế nên được vẽ dưới dạng biểu đồ cho mỗi hành động
- Tạo kịch bản cho
  - a basic flow,
  - một kịch bản cho mỗi luồng thay thế,
  - và một số kết hợp hợp lý của các luồng thay thế
- Tạo vòng lặp vô hạn



28

## 2.3. Các kỹ thuật kiểm thử hộp trắng

❖ Các trường hợp kiểm thử phải bao gồm tất cả cấu trúc xử lý trong mã nguồn

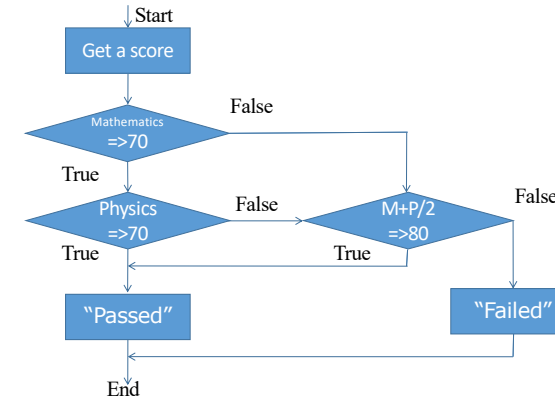
=> Phễu kiểm thử điển hình

- C0 measure: Các câu lệnh đã thực thi # / tất cả các câu lệnh #
  - C0 đo ở mức 100% có nghĩa là "tất cả các câu lệnh được thực hiện"
- C1 measure: Các nhánh vượt qua # / tất cả các nhánh #
  - C1 đo ở mức 100% có nghĩa là "tất cả các nhánh được thực hiện"

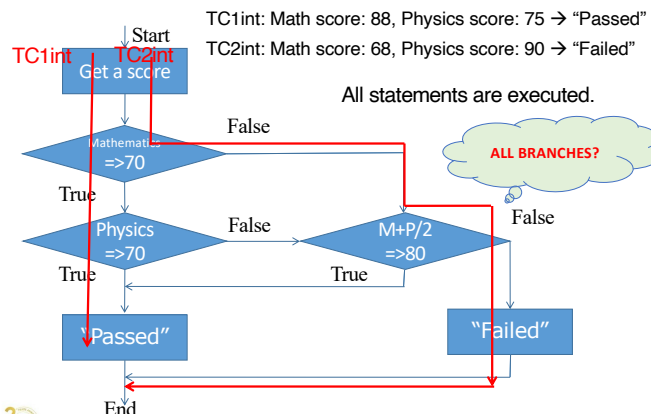
=> Ngăn các câu lệnh / nhánh không được để lại như các phần không được kiểm tra

=> Không thể phát hiện các chức năng không được triển khai

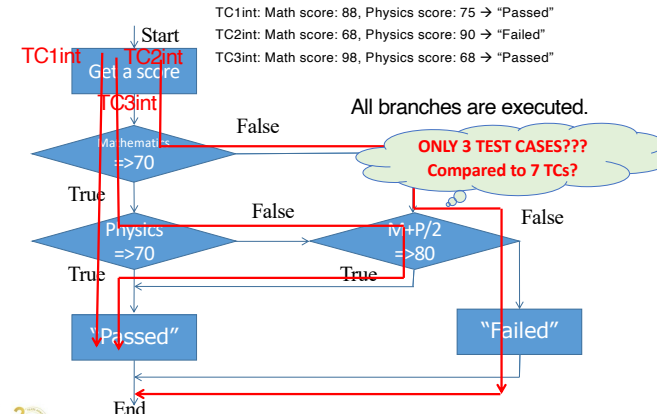
## VD. Kiểm thử luồng điều khiển cho "Examination Judgment Program"



## VD. Kiểm thử luồng điều khiển cho "Examination Judgment Program" – 100% C0 coverage



## VD. Kiểm thử luồng điều khiển cho "Examination Judgment Program" – 100% C1 coverage





### Bảng quyết định cho “Examination Judgment”

Condition1: Mathematics score  $\Rightarrow$  70  
 Condition2: Physics score  $\Rightarrow$  70  
 Condition3: Average of Mathematics, and Physics  $\Rightarrow$  80

	TC5	TC4	TC3	TC6	TC2	TC1	TCNG	TC7	
Condition1	True	True	True	True	False	False	False	False	False
Condition2	True	True	False	False	True	True	False	False	False
Condition3	True	False	True	False	True	False	True (none)	False	False
"Passed"	Yes	Yes	Yes	—	Yes	—	N/A	—	—
"Failed"	—	—	—	Yes	—	Yes	N/A	Yes	Yes

- ❖ One TCxint can cover plural TCs, based on the correct control flow structure
  - TC1int covers TC5 and TC4
  - TC2int covers TC1 and TC7
  - TC3int covers to TC3.
- ❖ TC2 and TC6 are left in no execution

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG 33

33

### VD. Kiểm thử luồng điều khiển cho “Examination Judgment Program” – 100% C1 coverage

TC2 is covered by TC2int and TC3int?  
 TC6 is covered by TC3int and TC2int?

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG 34

34

### VD. Kiểm thử luồng điều khiển cho “Examination Judgment Program” – 100% C1 coverage

Mistake ???  $\Rightarrow$  TC1int, TC2int and TC3int enough?  
 Only TC5 can't detect them  $\Rightarrow$  Both TC4, TC5 are needed

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG 35

35

### Kiểm thử đường dẫn dữ liệu / thông điệp để kiểm tra tích hợp

- ❖ Thực hiện kiểm thử hộp trắng bằng cách sử dụng biểu đồ tuần tự để kiểm tra tích hợp.
- $\Rightarrow$  Thực thi mọi đường dẫn / luồng thông điệp
- $\Rightarrow$  100% message path/flow coverage
- ❖ Có thể áp dụng cho dữ liệu / đường dẫn thông điệp / lưu đồ hoặc sơ đồ khác

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG 36

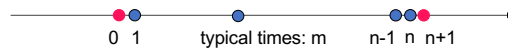
36

### Cách kiểm tra chương trình cấu trúc vòng lặp

❖ Đối với kiểm tra luồng điều khiển trong phần mềm bao gồm một vòng lặp, các tiêu chí sau thường được áp dụng thay vì các độ đo bao phủ C0 / C1..

- Skip the loop.
- Only one pass through the loop.
- Typical times m passes through the loop
- n, n-1, n+1 passes through the loop
  - n is maximum number, m is typical number ( $m < n$ )

❖ Ví dụ: 6 trường hợp dựa trên phân tích giá trị biên:



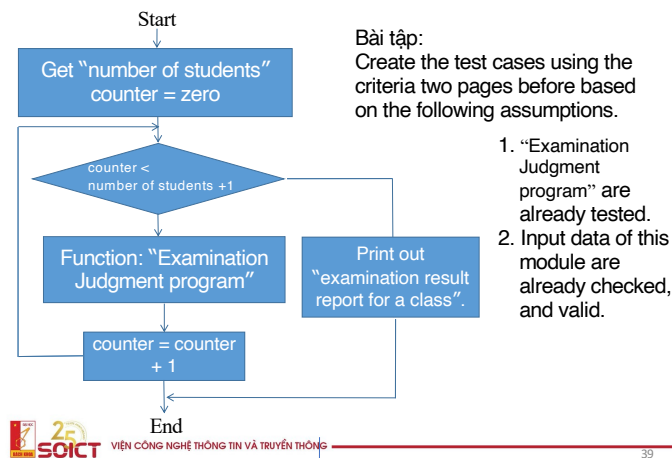
37

### Các ví dụ cho "Examination Judgment Program"

- Nhập điểm hai môn Toán và Vật lý cho mỗi thành viên của một lớp.
  - input form là dạng "tabular form".
  - Các thành viên trong lớp chỉ có thể được phép từ 0 (không) đến 50.
- Xuất / In ra "Báo cáo kết quả thi của một lớp".
  - Biểu mẫu đầu ra cũng là "dạng bảng" có các cột như tên học sinh, điểm số (Toán, Lý), đạt hay không đạt.

38

### Các ví dụ cho "Examination Judgment Program"

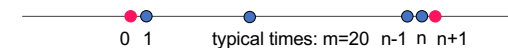


39

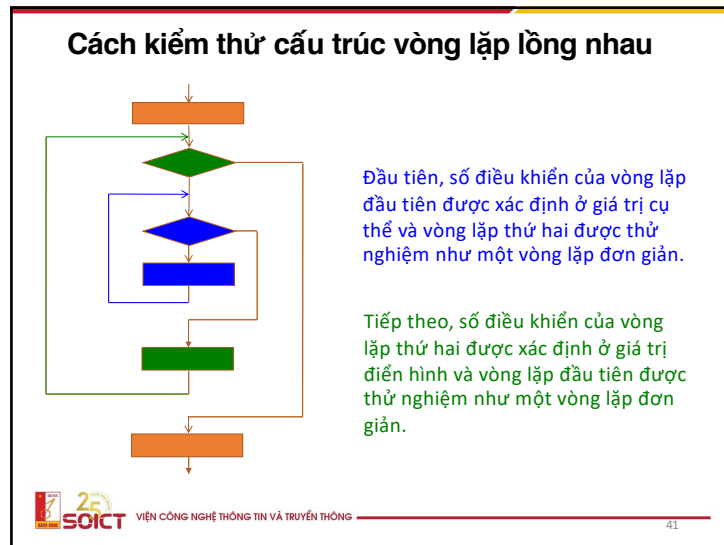
### Các ví dụ cho "Examination Judgment Program"

Loop test cases of the module are;  $n = 50$ .

- "number of students" = 0,
- "number of students" = 1,
- "number of students" = 20,
- "number of students" = 49,
- "number of students" = 50,
- "number of students" = 51 → Invalid.



40



41

### 2.4. Kết hợp Black/White Box test

- ❖ Ưu điểm của Black box
  - Kiểm tra bao gồm dựa trên đặc điểm kỹ thuật bên ngoài
  - Rất mạnh mẽ và cơ bản để phát triển phần mềm chất lượng cao
- ❖ Ưu điểm của White box
  - Nếu bất kỳ đường dẫn / luồng nào không xuất hiện trong các thông số kỹ thuật đã viết, các đường dẫn / luồng có thể bị bỏ sót trong các bài kiểm thử bao gồm => White box test
    - cho dữ liệu của hơn hai năm trước => đường dẫn thay thế
    - "0 <= score <= 100" => code: "if 0 <= score" and "if score <= 100"

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

42

### Cách thực hiện kiểm tra hiệu quả và đầy đủ

- ❖ Đầu tiên, thực hiện các bài kiểm tra dựa trên các thông số kỹ thuật bên ngoài
  - Nếu tất cả các trường hợp thử nghiệm đều thành công
  - => Tất cả các thông số kỹ thuật bên ngoài được thực hiện chính xác
- ❖ Thứ hai, thực hiện các bài kiểm tra dựa trên các thông số kỹ thuật bên trong
  - Thêm các trường hợp thử nghiệm để thực thi các đường dẫn / luồng còn lại, trong các thông số kỹ thuật bên ngoài
  - Nếu tất cả các trường hợp thử nghiệm đều thành công với mức độ phù hợp = 100%
  - => Tất cả các chức năng được chỉ định trong thông số kỹ thuật bên ngoài được thực hiện thành công mà không có bất kỳ mã dư thừa nào

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

43

### 2.5. JUnit

- ❖ Một công cụ test-driven development (junit.org)
- ❖ JUnit test generators hiện là một phần của nhiều Java IDEs (Eclipse, BlueJ, Jbuilder, DrJava)
- ❖ Các công cụ XUnit kể từ đó đã được phát triển cho nhiều ngôn ngữ khác (Perl, C ++, Python, Visual Basic, C #,...)

SOICT VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

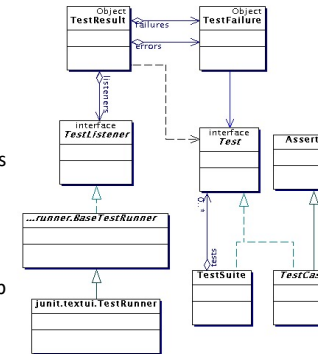
44

## Tại sao phải tạo một bộ thử nghiệm?

- ❖ Rõ ràng là bạn phải kiểm tra mã của mình — phải không?
  - Bạn có thể thực hiện thử nghiệm đột xuất (chạy bất kỳ thử nghiệm nào xảy ra với bạn vào lúc này) hoặc
  - Bạn có thể xây dựng một bộ thử nghiệm (một bộ thử nghiệm kỹ lưỡng có thể chạy bất cứ lúc nào)
- ❖ Nhược điểm của bộ thử nghiệm (test suite)
  - Phải lập trình thêm nhiều thứ
    - True, but use of a good test framework can help quite a bit
  - Bạn không có thời gian để làm thêm tất cả những công việc đó
    - False! Các thử nghiệm lặp đi lặp lại cho thấy rằng các bộ thử nghiệm giảm thời gian gỡ lỗi nhiều hơn so với số tiền dành để xây dựng bộ thử nghiệm
- ❖ Ưu điểm của bộ thử nghiệm
  - Giảm tổng số lỗi trong mã đã phân phối
  - Làm cho mã dễ bảo trì và dễ tái cấu trúc hơn nhiều

## Tổng quan kiến trúc

- ❖ Khung kiểm thử JUnit là một gói các lớp cho phép bạn viết các bài kiểm thử cho từng phương thức, sau đó dễ dàng chạy các bài kiểm tra đó
- ❖ **TestRunner** thực thi các kiểm thử và xuất báo cáo **TestResults**
- ❖ Bạn kiểm thử một lớp bằng cách mở rộng lớp ch trừu tượng **TestCase**
- ❖ Để viết các trường hợp kiểm thử, bạn cần biết và hiểu về lớp **Assert** class



## Viết một TestCase

- ❖ Để bắt đầu sử dụng JUnit, hãy tạo một lớp con của TestCase, lớp này bạn thêm các phương thức kiểm tra
- ❖ Đây là khung lớp kiểm thử:

```
import junit.framework.TestCase;
public class TestBowl extends TestCase {

    } //Test my class Bowl
```
- ❖ Tên của lớp rất quan trọng— nên có dạng là **TestMyClass** hoặc **MyClassTest**
- ❖ Quy ước đặt tên này giúp TestRunner tự động tìm các lớp kiểm thử của bạn

## Viết các phương thức trong TestCase

- ❖ Mẫu dưới đây, mô thức lập trình theo hợp đồng **programming by contract** paradigm:
  - Thiết lập **preconditions**
  - Exercise functionality being tested
  - Kiểm tra **postconditions**
- ❖ Ví dụ:

```
public void testEmptyList() {
    Bowl emptyBowl = new Bowl();
    assertEquals("Size of an empty list should be zero.",
        0, emptyList.size());

    assertTrue("An empty bowl should report empty.",
        emptyBowl.isEmpty());
}
```
- ❖ Những điều cần lưu ý:
  - Chữ ký phương thức xác định — public void **testWhatever()**
    - Cho phép chúng được tìm thấy và gọi tự động bởi JUnit
  - Viết mã theo mẫu
  - Chú ý các lời gọi assert-type calls...

## Các phương thức Assert

- ❖ Các phương thức Assert methods có các tham số dạng như sau:  
*message, expected-value, actual-value*
- ❖ Floating point numbers get an additional argument, a tolerance
- ❖ Mỗi phương thức assert có một phiên bản tương đương không nhận thông báo - tuy nhiên, việc sử dụng này không được khuyến khích vì:
  - Các thông điệp giúp ghi nhận các kiểm thử
  - Các thông điệp cung cấp thông tin bổ sung khi đọc nhật ký lỗi

## Các phương thức Assert

- ❖ `assertTrue(String message, Boolean test)`
- ❖ `assertFalse(String message, Boolean test)`
- ❖ `assertNull(String message, Object object)`
- ❖ `assertNotNull(String message, Object object)`
- ❖ `assertEquals(String message, Object expected, Object actual)`  
(uses equals method)
- ❖ `assertSame(String message, Object expected, Object actual)`  
(uses == operator)
- ❖ `assertNotSame(String message, Object expected, Object actual)`

## Nhiều thứ hơn trong các lớp thử nghiệm

- ❖ Giả sử bạn muốn kiểm thử một lớp `Counter`
- ❖ `public class CounterTest`  
`extends junit.framework.TestCase {`
  - This is the unit test for the `Counter` class
- ❖ `public CounterTest() { } //Default constructor`
- ❖ `protected void setUp()`
  - Test *fixture* creates and initializes instance variables, etc.
- ❖ `protected void tearDown()`
  - Releases any system resources used by the test fixture
- ❖ `public void testIncrement(), public void testDecrement()`
  - These methods contain tests for the `Counter` methods `increment()`, `decrement()`, etc.
  - Note capitalization convention

## JUnit tests for Counter

```
public class CounterTest extends junit.framework.TestCase {
    Counter counter1;
    public CounterTest() { } // default constructor

    protected void setUp() { // creates a (simple) test fixture
        counter1 = new Counter();
    }

    public void testIncrement() {
        assertTrue(counter1.increment() == 1);
        assertTrue(counter1.increment() == 2);
    }

    public void testDecrement() {
        assertTrue(counter1.decrement() == -1);
    }
}
```

Lưu ý rằng mỗi bài kiểm thử bắt đầu với một bộ đếm hoàn toàn mới

Điểm này có nghĩa là bạn không phải lo lắng về thứ tự chạy các bài kiểm thử

## TestSuites

- ❖ TestSuites tập hợp các bài kiểm thử đã lựa chọn để chạy chúng như một đơn vị
- ❖ Các bộ sưu tập tự động sử dụng TestSuites, tuy nhiên, để chỉ định thứ tự chạy các bài kiểm thử, hãy viết :

```
public static Test suite() {
    suite.addTest(new TestBowl("testBowl"));
    suite.addTest(new TestBowl("testAdding"));
    return suite;
}
```

- ❖ Ít khi phải viết TestSuites của riêng bạn vì mỗi phương thức trong TestCase của bạn phải độc lập với tất cả các phương thức khác
- ❖ Can create TestSuites that test a whole package:

```
public static Test suite() {
    TestSuite suite = new TestSuite();
    suite.addTestSuite(TestBowl.class);
    suite.addTestSuite(TestFruit.class);
    return suite;
}
```

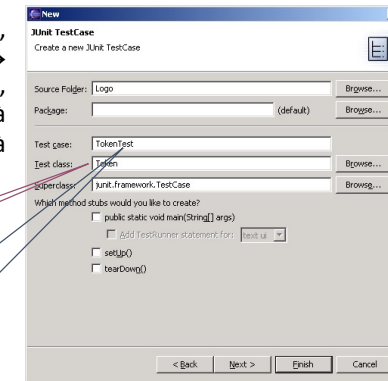
53

## JUnit in Eclipse

- ❖ Để tạo một test mới, chọn File → New → Other... → Java, JUnit, TestCase và nhập tên của lớp mà bạn sẽ test

Fill this in

This will be filled in automatically



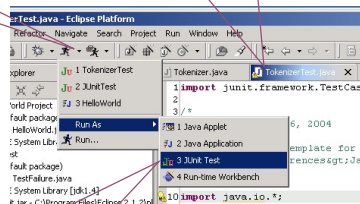
54

## Thực thi JUnit

Second, use this  
pull-down menu

First, select a Test class

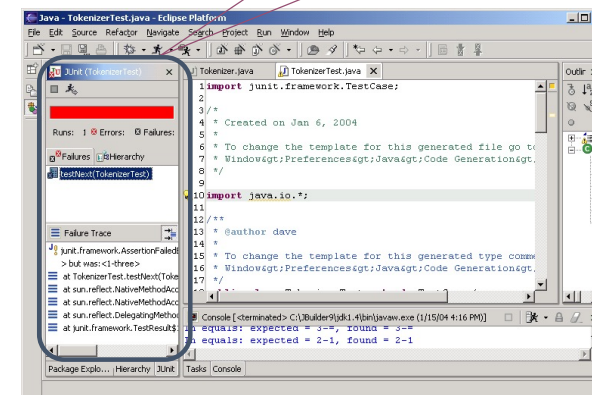
Third, Run As → JUnit Test



55

## Kết quả

Your results are here



56

## Kiểm thử đơn vị cho các ngôn ngữ khác

- ❖ Các công cụ kiểm thử đơn vị phân biệt giữa :
  - Errors (unanticipated problems caught by exceptions)
  - Failures (anticipated problems checked with assertions)
- ❖ Đơn vị cơ bản của kiểm thử:
  - `CPPUNIT_ASSERT(Bool)` kiểm tra một biểu thức
- ❖ CppUnit có nhiều lớp kiểm thử (e.g. *TestFixture*)
  - Kế thừa chúng và nạp chồng các phương thức

57

## Một ví dụ khác: sqrt

```
// throws: IllegalArgumentException if x < 0
// returns: approximation to square root of x
public double sqrt(double x)
```

### Một số giá trị hoặc phạm vi của x có thể đáng kiểm thử là gì

- ❑  $x < 0$  (exception thrown)
- ❑  $x \geq 0$  (returns normally)
- ❑ around  $x = 0$  (boundary condition)
- ❑ perfect squares (`sqrt(x)` an integer), non-perfect squares
- ❑  $x < \text{sqrt}(x)$ ,  $x > \text{sqrt}(x)$
- ❑ Specific tests: say  $x = \{-1, 0, 0.5, 1, 4\}$

58

## Các miền con

- ❖ Nhiều lần thực thi phản ánh cùng một hành vi - ví dụ: đối với **sqrt**, kỳ vọng là
  - Tất cả  $x < 0$  đầu vào sẽ ném ra một ngoại lệ
  - Tất cả  $x \geq 0$  đầu vào sẽ trả về bình thường với một câu trả lời đúng
- ❖ Bằng cách thử nghiệm bất kỳ phần tử nào từ mỗi **miền con**, mục đích là để thử nghiệm đơn lẻ đại diện cho các hành vi khác của tên miền phụ - mà *không cần thử nghiệm chúng!*
- ❖ Tất nhiên, điều này không dễ dàng như vậy - ngay cả trong ví dụ đơn giản ở trên, còn khi  $x$  tràn?

59

## Kiểm thử RandomHello

- ❖ “Tạo lớp Java đầu tiên của bạn với phương thức main sẽ chọn giá trị ngẫu nhiên, rồi in ra console, một trong năm lời chào có thể có mà bạn định nghĩa.”
- ❖ Chúng ta sẽ tập trung vào phương thức **getGreeting**, trả về ngẫu nhiên một trong năm lời chào
- ❖ Chúng ta sẽ tập trung vào *black-box testing* – chúng tôi sẽ làm việc mà không có kiến thức về việc triển khai
- ❖ Và chúng ta sẽ tập trung vào kiểm thử đơn vị bằng cách sử dụng JUnit framework
- ❖ Trộn lẫn, với bất kỳ sự may mắn nào, trang trình bày và bản trình diễn

60

## Có thực thi và trả về kết quả?

- ❖ Nếu `getGreeting` không chạy và trả về mà không ném ra một ngoại lệ, nó không thể đáp ứng đặc tả

JUnit tag "this is a test"	<code>@Test</code>
name of test	<code>public void test_NoException() {</code>
Run <code>getGreeting</code>	<code>RandomHello.getGreeting();</code>
JUnit "test passed" (doesn't execute if exception thrown)	<code>assertTrue(true);</code> <code>}</code>

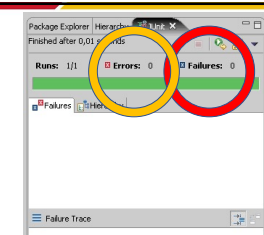
Tests should have descriptive (often very long) names

A unit test is a (stylized) program!  
When you're writing unit tests (and many other tests), you're programming!

61

## Thực thi JUnit tests

- ❖ Có nhiều cách để chạy phương thức kiểm thử JUnit, các lớp kiểm thử và bộ kiểm thử
- ❖ Tổng quát, chọn method, class hoặc suite và Run As >> JUnit Test
- ❖ Thanh màu xanh cho "all tests pass"
- ❖ Thanh màu đỏ cho ít nhất một test failed hoặc đã có error
- ❖ The failure trace cho biết test nào thất bại và tại sao



- A failure is when the test doesn't pass – that is, the oracle it computes is incorrect
- An error is when something goes wrong with the program that the test didn't check for (e.g., a null pointer exception)

62

## Có trả lại một trong những lời chào không?

- ❖ Nếu không trả về một trong những lời chào đã xác định, nó không thể đáp ứng đặc tả

```
@Test
public void testDoes_getGreeting_returnDefinedGreeting() {
    String rg = RandomHello.getGreeting();
    for (String s : RandomHello.greetings) {
        if (rg.equals(s)) {
            assertTrue(true);
            return;
        }
    }
    fail("Returned greeting not in greetings array");
}
```

63

## A JUnit test class

```
import org.junit.*;
import static org.junit.Assert.*;

public class RandomHelloTest() {
    @Test
    public void test_ReturnDefinedGreeting() {
        ...
    }
    @Test
    public void test_EveryGreetingReturned() {
        ...
    }
    ...
}
```

Don't forget that Eclipse can help you get the right import statements – use Organize Imports (Ctrl-Shift-O)

- ❑ All `@Test` methods run when the test class is run
- ❑ That is, a JUnit test class is a set of tests (methods) that share a (class) name

64



## Có trả về một lời chào ngẫu nhiên?

```
@Test
public void testDoes_getGreetingNeverReturnSomeGreeting() {
    int greetingCount = RandomHello.greetings.length;
    int count[] = new int[greetingCount];
    for (int c = 0; c < greetingCount; c++)
        count[c] = 0;
    for (int i = 1; i < 100; i++) {
        String rs = RandomHello.getGreeting();
        for (int j = 0; j < greetingCount; j++)
            if (rs.equals(RandomHello.greetings[j]))
                count[j]++;
    }
    for (int j = 0; j < greetingCount; j++)
        if (count[j] == 0)
            fail(j+"th [0-4] greeting never returned");
    assertTrue(true);
}
```

Run it 100 times

If even one greeting is never returned, it's unlikely to be random (~1-0.8<sup>100</sup>)

65

## Còn về một nhà phát triển nhếch nhác?

```
if (randomGenerator.nextInt(2) == 0) {
    return (greetings[0]);
} else
    return (greetings[randomGenerator.nextInt(5)]);
```

- ❑ Lật đồng xu và chọn một lời chào ngẫu nhiên hoặc một lời chào cụ thể
- ❑ Trước đó "có phải là ngẫu nhiên không?" kiểm thử hầu như sẽ luôn vượt qua khi triển khai này
- ❑ Nhưng nó không đáp ứng đặc tả, vì nó không phải là một lựa chọn ngẫu nhiên

66

## Thay vì thế: Sử dụng thống kê đơn giản

```
@Test
public void test_UniformGreetingDistribution() {
    // ...count frequencies of messages returned, as in
    // ...previous test (test_EveryGreetingReturned)

    float chiSquared = 0f;
    float expected = 20f;
    for (int i = 0; i < greetingCount; i++)
        chiSquared = chiSquared +
            ((count[i]-expected)*
             (count[i]-expected))
            /expected;
    if (chiSquared > 13.277) // df 4, pvalue .01
        fail("Too much variance");
}
```

67

## A JUnit test suite

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    RandomHelloTest.class,
    SleazyRandomHelloTest.class
})
public class AllTests {
    // this class remains completely
    // empty, being used only as a
    // holder for the above
    // annotations
}
```

- ❑ Xác định một bộ cho mỗi chương trình (hiện tại)
- ❑ The suite cho phép nhiều lớp kiểm thử— each of which has its own set of `@Test` methods – to be defined and run together
- ❑ Add `tc.class` to the `@Suite.SuiteClasses` annotation if you add a new test class named `tc`
- ❑ Vì vậy, bộ kiểm thử JUnit là một tập hợp các lớp kiểm thử (khiến nó trở thành một tập hợp các phương thức kiểm thử)

68

## Các phương thức JUnit assertion

...khiến kiểm thử hiện tại không thành công...

fail()	immediately
assertTrue(tst)	if tst is false
assertFalse(tst)	if test is true
assertEquals(expected, actual)	if expected does not equal actual
assertSame(expected, actual)	if expected != actual
assertNotSame(expected, actual)	if oracle == actual
assertNotNull(value)	if value is not null
assertNotNull(value)	if value is null

- ❖ Có thể thêm thông báo lỗi: `assertNotNull("Ptr isn't null", value)`
- ❖ **expected** là giá trị tiên tri- hãy nhớ đây là tham số đầu tiên (ngoài cùng bên trái)
- ❖ Bảng trên chỉ mô tả khi nào thất bại - điều gì xảy ra nếu một khẳng định thành công? Bài kiểm tra có vượt qua không?

## ArrayList: các ví dụ kiểm thử

```
@Test
public void testAddGet1() {
    ArrayList list = new
        ArrayList();
    list.add(42);
    list.add(-3);
    list.add(15);
    assertEquals(42, list.get(0));
    assertEquals(-3, list.get(1));
    assertEquals(15, list.get(2));
}
```

```
@Test
public void testIsEmpty() {
    ArrayList list = new
        ArrayList();
    assertTrue(list.isEmpty());
    list.add(123);
    assertFalse(list.isEmpty());
    list.remove(0);
    assertTrue(list.isEmpty());
}
```

- ❑ Khái niệm cấp cao: kiểm tra hành vi kết hợp
  - ❑ Có thể **add** hoạt động khi được gọi một lần, nhưng không hoạt động khi được gọi hai lần
  - ❑ Có thể **add** tự hoạt động, nhưng thất bại (hoặc gây ra lỗi) sau khi gọi **remove**

## Một vài gợi ý: cấu trúc dữ liệu

- ❖ Cần phải truyền nhiều mảng? Sử dụng array literals

```
public void exampleMethod(int[] values) { ... }
...
exampleMethod(new int[] {1, 2, 3, 4});
exampleMethod(new int[] {5, 6, 7});
```

- ❖ Cần một **ArrayList** nhanh?

```
List<Integer> list = Arrays.asList(7, 4, -2, 3, 9, 18);
```

- ❖ Cần một set, queue, etc. nhanh? Many take a list

```
Set<Integer> list = new HashSet<Integer>()
    Arrays.asList(7, 4, -2, 9));
```

## Một vài gợi ý chung

- ❖ Thử nghiệm từng thứ một trong mỗi phương thức thử nghiệm
  - 10 bài kiểm thử nhỏ tốt hơn nhiều so với một bài kiểm thử lớn
- ❖ Be stingy with **assert** statements
  - Lệnh **assert** đầu tiên thất bại sẽ dừng kiểm thử- không cung cấp thông tin về việc liệu **assert** sau này có thất bại hay không
- ❖ Be stingy with logic
  - Tránh **try/catch** - nếu nó được đưa ra một ngoại lệ, hãy sử dụng **expected=** ... Nếu không, hãy để JUnit bắt nó

## Trường hợp kiểm thử nguy hiểm

### ❖ Thứ tự kiểm thử phụ thuộc

- Nếu chạy Kiểm thử A trước Kiểm thử B cho kết quả khác với việc chạy Kiểm thử B rồi đến Kiểm thử A, thì có thể có điều gì đó khó hiểu và cần được trình bày rõ ràng

### ❖ Trạng thái chia sẻ có thể thay đổi

- Tests A và B cả hai đều sử dụng một đối tượng được chia sẻ – nếu A breaks the object, điều gì xảy ra với B?
  - Đây là một dạng của lệnh kiểm thử phụ thuộc
  - We will explicitly talk about invariants over data representations and testing if the invariants are ever broken

73

## More JUnit

### ❖ Hết giờ - không muốn đợi mãi để kiểm thử hoàn thành

### ❖ Kiểm thử các ngoại lệ

```
@Test(expected =  
    ArrayIndexOutOfBoundsException.class)  
public void testBadIndex() {  
    ArrayList list = new ArrayList();  
    list.get(4); // this should raise the exception  
} // and thus the test will pass
```

### ❖ Setup [teardown] – methods to run before [after] each test case method [test class] is called

74

## Nội dung

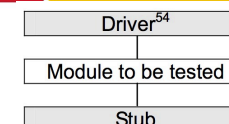
### 1. Tổng quan về kiểm thử

### 2. Kiểm thử đơn vị

### ➔ 3. Kiểm thử tích hợp

75

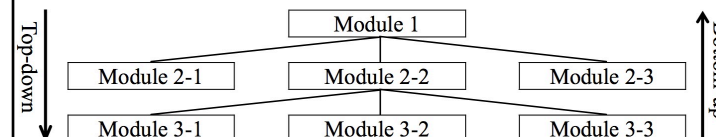
## 3. Kiểm thử tích hợp



### ❖ Kiểm tra giao diện giữa các mô-đun cũng như đầu vào và đầu ra

### ❖ Stub/Driver:

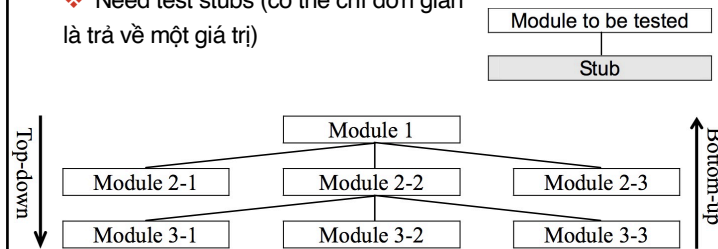
- Một chương trình mô phỏng các chức năng của mô-đun cấp thấp hơn / cấp cao hơn



76

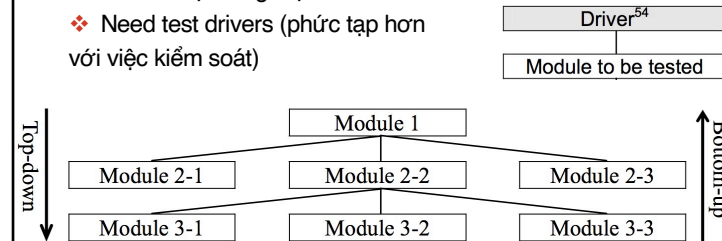
### 3.1. Cách tiếp cận từ trên xuống

- ❖ Có thể phát hiện sớm các khiếm khuyết do hiểu sai đặc tả
- ❖ Hiệu quả trong các hệ thống mới phát triển
- ❖ Need test stubs (có thể chỉ đơn giản là trả về một giá trị)



### 3.2. Cách tiếp cận từ dưới lên

- ❖ Các mô-đun thấp hơn là độc lập => kiểm tra độc lập và song song
- ❖ Hiệu quả trong việc phát triển hệ thống bằng cách sửa đổi các hệ thống hiện có
- ❖ Need test drivers (phức tạp hơn với việc kiểm soát)



### 3.3. Các kỹ thuật kiểm thử tích hợp khác

- ❖ Kiểm thử Big-bang
  - Trong đó tất cả các mô-đun đã hoàn thành các bài kiểm tra đơn vị được liên kết cùng một lúc và được kiểm tra
  - Giảm số lượng các thủ tục thử nghiệm trong chương trình quy mô nhỏ; nhưng không dễ xác định lỗi
- ❖ Kiểm thử Sandwich
  - Trong đó các mô-đun cấp thấp hơn được kiểm tra từ dưới lên và các mô-đun cấp cao hơn được kiểm tra từ trên xuống

### 3.4. Kiểm thử hồi quy

*“Khi bạn sửa một lỗi, bạn sẽ tạo ra một số lỗi mới”*

- ❖ Kiểm thử lại một ứng dụng sau khi mã của nó đã được sửa đổi để xác minh rằng nó vẫn hoạt động chính xác
  - Chạy lại các trường hợp kiểm thử hiện có
  - Kiểm tra để đảm bảo rằng các thay đổi mã không phá vỡ bất kỳ chức năng hoạt động nào trước đó (tác dụng phụ)(side-effect)
- ❖ Chạy thường xuyên nhất có thể
- ❖ Với công cụ kiểm thử hồi quy tự động



81



82