**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**

**KHOA CÔNG NGHỆ THÔNG TIN**

**======\*\*\*======**

****

**TX1 KIỂM THỬ PHẦN MỀM**

|  |  |
| --- | --- |
| GVHD: | Ths. Hoàng Quang Huy |
| Nhóm - Lớp: | Nhóm 7\_20212IT6013004 |
| Họ và tên: | Trần Thái Quyền |
| Mã sinh viên | 2019605210 |

Hà nội, Năm 2022

Câu 1: Kiểm thử hộp trắng

1. **Kiểm thử hộp trắng là gì ?**

Kiểm thử hộp trắng là kiểm thử dựa vào cấu trúc mã lệnh chương trình. Phương pháp kiểm thử hộp trắng (một phần chương trình, hay một hệ thống, một phần của hệ thống) đáp ứng tốt tất cả các giá trị input bao gồm cả các giá trị không đúng hay không theo dự định của chương trình.

Kiểm thử hộp trắng bao gồm phân tích dòng dữ liệu , điều khiển dòng, dòng thông tin, mã thực hành, ngoại lệ và những lỗi trình bày trong hệ thống để kiểm tra những hành động của phần mềm không được định trước.

Kiểm thử hộp trắng còn là một trong hai phương pháp kiểm thử Box Testing trong kiểm thử phần mềm. Mặt khác, kiểm thử hộp trắng trong kỹ thuật phần mềm dựa trên hoạt động bên trong của một ứng dụng và xoay quanh kiểm thử nội bộ.

INPUT OUTPUT

1. **Làm thế nào để thực hiện kiểm thử hộp trắng?**

**Bước 1:** Hiểu mã nguồn

Điều đầu tiên mà người kiểm thử sẽ thường làm là học và hiểu mã nguồn của ứng dụng. Vì kiểm thử hộp trắng liên quan đến việc kiểm tra hoạt động bên trong của một ứng dụng, người kiểm thử phải rất hiểu biết về ngôn ngữ lập trình được sử dụng trong các ứng dụng mà họ đang kiểm tra. Ngoài ra, người kiểm thử phải có nhận thức cao về các phương pháp mã hóa an toàn để tìm ra các vấn đề bảo mật và ngăn chặn các cuộc tấn công từ tin tặc cố ý hoặc vô tình đưa mã độc vào ứng dụng.

**Bước 2:** Tạo các trường hợp kiểm thử và thực hiện

Kiểm thử mã nguồn của ứng dụng để tìm luồng và cấu trúc phù hợp. Một cách là viết thêm mã lệnh để kiểm tra mã nguồn của ứng dụng. Người kiểm thử sẽ phát triển các bài kiểm tra nhỏ cho từng quy trình hoặc chuỗi quy trình trong ứng dụng. Phương pháp này yêu cầu người kiểm thử phải có kiến ​​thức sâu sắc về mã lệnh và thường được thực hiện bởi nhà phát triển. Các phương pháp khác bao gồm kiểm thử thủ công, thử nghiệm và kiểm tra lỗi và sử dụng các công cụ kiểm thử.

1. **Một số kỹ thuật kiểm thử hộp trắng**

* Kỹ thuật bao phủ lệnh (Statement Coverage)
* Kỹ thuật bao phủ quyết định (Decision Coverage)
* Kỹ thuật bao phủ nhánh (Branch Coverage)
* Kỹ thuật bao phủ điều kiện (Condition Coverage)
* Kỹ thuật bao phủ đường dẫn (Path Coverage)

1. **Công cụ kiểm thử hộp trắng**

Dưới đây là danh sách các công cụ kiểm tra hộp trắng hàng đầu:

* Parasoft Jtest
* EclEmma
* NUnit
* PyUnit
* HTMLUnit
* CppUnit

1. **Ưu điểm của kiểm thử hộp trắng**

* Test có thể bắt đầu ở giai đoạn sớm hơn, ngay cả khi giao diện người dùng không khả dụng
* Test kỹ càng hơn, có thể bao phủ hầu hết các đường dẫn.
* Cho phép tìm kiếm các lỗi ẩn bên trong
* Các lập trình viên có thể tự kiểm tra
* Giúp tối ưu việc mã hoá
* Do yêu cầu kiến thức cấu trúc bên trong của phần mềm, nên việc kiểm soát lỗi tối đa nhất.
* Thích hợp trong việc tìm kiếm lỗi và các vấn đề trong mã lệnh.

1. **Nhược điểm của kiểm thử hộp trắng**

* Kiểm thử hộp trắng có thể khá phức tạp nên tốn nhiều chi phí và thời gian.
* Các nhà phát triển thường thực hiện các trường hợp kiểm thử hộp trắng sẽ ghét nó. Việc kiểm tra hộp trắng của các nhà phát triển không chi tiết có thể dẫn đến lỗi sản xuất.
* Kiểm thử hộp trắng đòi hỏi nguồn lực chuyên nghiệp, hiểu biết chi tiết về lập trình và triển khai.

**Câu 2: Kỹ thuật bao phủ lệnh (Statement Coverage)**

Statement Coverage Testing là một trong những phần mềm kiểm thử được sử dụng rộng rãi. Nó đi kèm với thử nghiệm hộp trắng.

Statement Coverage Testing được sử dụng để thiết kế các trường hợp kiểm thử hộp trắng. Kỹ thuật này liên quan đến việc thực thi tất cả các câu lệnh của mã nguồn ít nhất một lần. Nó được sử dụng để tính tổng số câu lệnh được thực thi trong mã nguồn trên tổng số câu lệnh có trong mã nguồn.

Mức độ bao phủ của câu lệnh tạo ra kịch bản về các trường hợp kiểm thử trong quy trình kiểm thử hộp trắng dựa trên cấu trúc của mã.

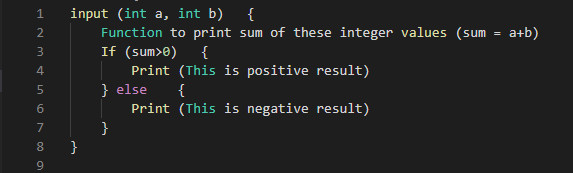
Statement coverage = \* 100

Trong thử nghiệm hộp trắng, người thử nghiệm tập trung vào hoạt động của mã nguồn nội bộ và lưu đồ hoặc biểu đồ luồng của mã.

Trong mã nguồn nội bộ, có rất nhiều phần tử như toán tử, phương thức, mảng, vòng lặp, câu lệnh điều khiển, trình xử lý ngoại lệ, … Dựa trên đầu vào được cung cấp cho chương trình, một số câu lệnh mã được thực thi và một số câu lệnh có thể không được thực thi. Mục tiêu của kỹ thuật bao hàm câu lệnh là bao hàm tất cả các câu lệnh đang thực thi có thể có và các đường dẫn trong mã.

**Ví dụ**

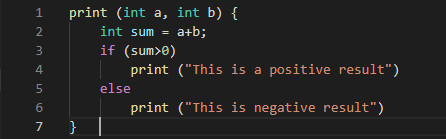
* Lấy đầu vào của hai giá trị như a = 0 và b = 1.
* Tìm tổng của hai giá trị này.
* Nếu tổng lớn hơn 0, thì in ra “Đây là kết quả dương.”
* Nếu tổng nhỏ hơn 0, thì in “Đây là kết quả âm”.



Có hai kịch bản khác nhau và tính toán phần trăm bảo hiểm báo cáo cho mã nguồn nhất định.

**Trường hợp 1:**

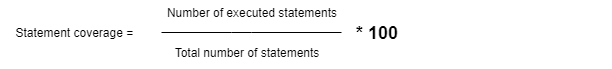
Nếu a = 5, b = 4



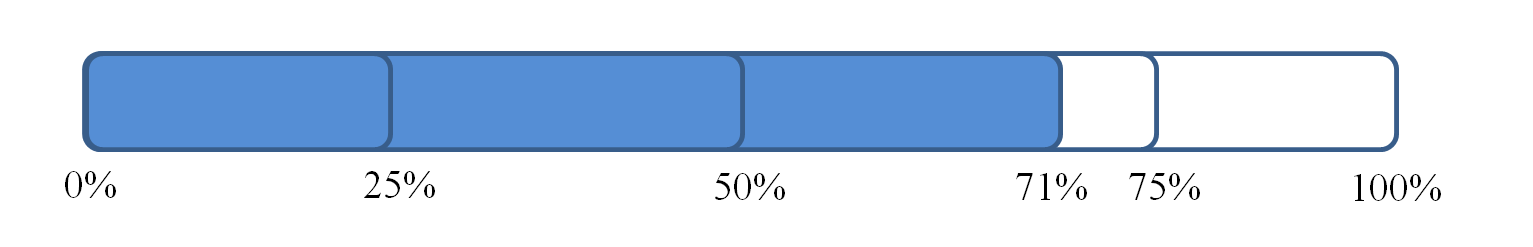
Trong kịch bản 1, chúng ta có thể thấy giá trị của tổng sẽ là 9 lớn hơn 0 và theo điều kiện kết quả sẽ là “Đây là một kết quả dương.” Các câu lệnh được tô màu đỏ là các câu lệnh được thực thi của kịch bản này.

Để tính toán mức độ phù hợp của câu lệnh của tình huống đầu tiên, hãy lấy tổng số câu lệnh là 7 và số câu lệnh được sử dụng là 5.

* Tổng số câu lệnh = 7
* Số câu lệnh được thực thi = 5

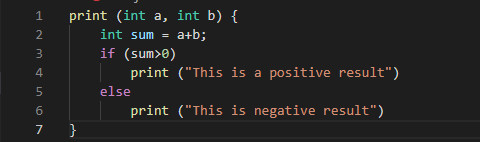


Statement coverage = \*100 = = 71%



**Trường hợp 2:**

Nếu A = -2, B = -7

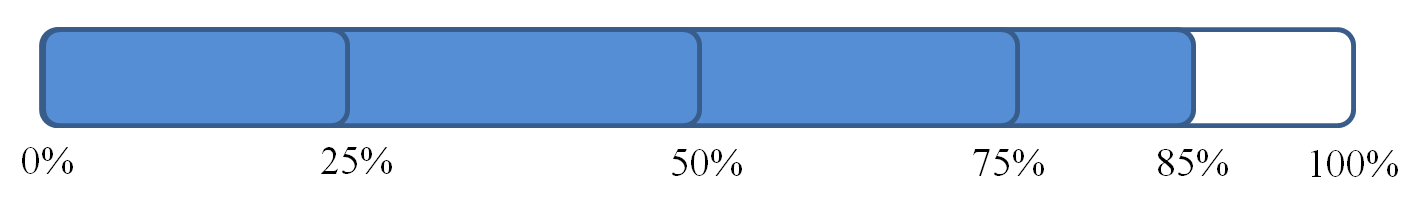


Trong trường hợp 2, giá trị của tổng sẽ là -9 nhỏ hơn 0 và theo điều kiện, kết quả sẽ là “Đây là kết quả âm”. Các câu lệnh được tô màu đỏ là các câu lệnh được thực thi của kịch bản này.

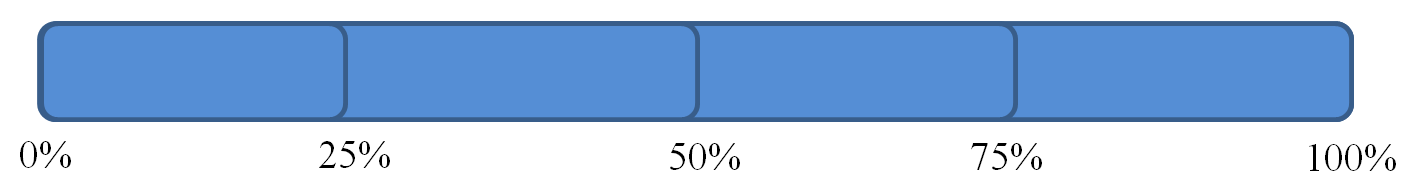
Để tính toán mức độ phù hợp của câu lệnh trong tình huống đầu tiên, hãy lấy tổng số câu lệnh là 7 và số câu lệnh được sử dụng là 6.

* Tổng số câu lệnh = 7
* Số câu lệnh đã thực thi = 6

Statement coverage = \*100 = = 85%



Tuy nhiên, chúng ta có thể thấy tất cả các tuyên bố đều được bao hàm trong cả hai kịch bản và chúng ta có thể coi rằng phạm vi báo cáo tổng thể là 100%.



**Câu 3:**

**Exercise 3.1**

Bài mẫu:

**public** **class** Employee {

String name;

**int** hoursWorked;

**float** hourlyRate;

**float** tax;

**float** grossPay;

**float** netPay;

**public** Employee(String name, **int** hoursWorked, **float** hourlyRate){

**this**.name = name;

**this**.hoursWorked = hoursWorked;

**this**.hourlyRate= hourlyRate;

}

**public** String getName(){

**return** name;

}

**public** **void** calculatePay(){

**int** payableHours;

**if**( hoursWorked <= 40){

payableHours = hoursWorked;

}**else**{

payableHours = 40+(hoursWorked - 40)\* 3/2;

}

grossPay = hoursWorked \* hourlyRate;

**if**(grossPay >= 200){

tax = grossPay \* 20/100;

}

netPay = grossPay - tax;

System.***out***.println(grossPay +"\t"+tax+"\t"+netPay);

}// end calculatePay()

}

Main

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Employee A=**new** Employee("ADAMS",35,5);

Employee B=**new** Employee("BAKER",40,5);

Employee C=**new** Employee("CAIRNS",44,5);

Employee D=**new** Employee("DONALD",20,6);

A.calculatePay();

B.calculatePay();

C.calculatePay();

D.calculatePay();

}

}

\*, kết quả chạy

175.0 0.0 175.0

200.0 40.0 160.0

220.0 44.0 176.0

120.0 0.0 120.0

\* code đã sửa

**public** **class** Employee {

String name;

**int** hoursWorked;

**float** hourlyRate;

**float** tax;

**float** grossPay;

**float** netPay;

**public** Employee(String name, **int** hoursWorked, **float** hourlyRate){

**this**.name = name;

**this**.hoursWorked = hoursWorked;

**this**.hourlyRate= hourlyRate;

}

**public** String getName(){

**return** name;

}

**public** **void** calculatePay(){

**int** payableHours;

**if**( hoursWorked <= 40){

payableHours = hoursWorked;

}**else**{

payableHours = 40+(hoursWorked - 40)\* 3/2;

}

grossPay = payableHours \* hourlyRate; // fixed

**if**(grossPay >= 200){ // fixed remove =

tax = (grossPay-200) \* 20/100; // fixed

}

netPay = grossPay - tax;

System.***out***.println(grossPay +"\t"+tax+"\t"+netPay);

}// end calculatePay()

}

Main

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Employee A=**new** Employee("ADAMS",35,5);

Employee B=**new** Employee("BAKER",40,5);

Employee C=**new** Employee("CAIRNS",44,5);

Employee D=**new** Employee("DONALD",20,6);

A.calculatePay();

B.calculatePay();

C.calculatePay();

D.calculatePay();

}

}

**Result**

175.0 0.0 175.0

200.0 0.0 200.0

230.0 6.0 224.0

120.0 0.0 120.0

**Exercise 3.2**

**Sample**

**package** ex2;

**class** Arrays {

String [] arrayA= {

"dog","cat","rat","fox","frog",

"dinosaur","cow","bull"

};

String [] arrayB= {

"dog", "dog", "cat","dog","frog","rat","turtle",

"fox","fox","man","man","frog","man"

};

**public** **void** compare(){

**int** counter =0;

**int** indexB = 0;

**for** (**int** indexA =0; indexA<arrayA.length;indexA++){

**for**(indexB=0; indexB<arrayB.length;indexB++){

**if**(arrayA[indexA] == arrayB[indexB]){

counter++;

}

}// end for

**if**(counter > 0){

System.***out***.println(arrayB[indexB -1] + " " + counter);

}

}// end for

}//end compare()

**public** **static** **void** main(String args[]){

Arrays a = **new** Arrays();

a.compare();

}

}// end class Arrays

**Main**

**package** ex2;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Arrays a = **new** Arrays();

a.compare();

}

}

**Result**

man 3

man 4

man 5

man 7

man 9

man 9

man 9

man 9

* **Fixed**

**package** ex2;

**class** Arrays {

String [] arrayA= {

"dog","cat","rat","fox","frog",

"dinosaur","cow","bull"

};

String [] arrayB= {

"dog", "dog", "cat","dog","frog","rat","turtle",

"fox","fox","man","man","frog","man"

};

**public** **void** compare(){

**int** counter =0;

**int** indexB = 0;

**for** (**int** indexA =0; indexA<arrayA.length;indexA++){

**for**(indexB=0; indexB<arrayB.length;indexB++){

**if**(arrayA[indexA] == arrayB[indexB]){

counter++;

}

}// end for

**if**(counter > 0){

System.***out***.println(arrayA[indexA] + " " + counter); // fixed

}

counter=0; //fixed

}// end for

}//end compare()

**public** **static** **void** main(String args[]){

Arrays a = **new** Arrays();

a.compare();

}

}// end class Arrays

**package** ex2;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

Arrays a = **new** Arrays();

a.compare();

}

}

**Result**

dog 3

cat 1

rat 1

fox 2

frog 2