|  |  |
| --- | --- |
| TRƯỜNG ĐẠI HỌC SÀI GÒN | **KỲ THI KẾT THÚC HỌC PHẦN** |
|  | **Học phần: Kiểm thử phần mềm**  **Mã học phần:   841050**  **Học Kỳ 2      Năm học 2020-2021** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Họ tên sinh viên :**Phạm Đăng Hoàng  **Ngày sinh**: 12/11/2000 | | | **Mã số sinh viên**  3118410133 | **Mã nhóm thi**  2004 |
| **Điểm** | | Họ tên (chữ ký) cán bộ chấm thi 1 | Họ tên (chữ ký) cán bộ chấm thi 2 | |
| *Bài tập* | *Phát vấn* |

Bài làm

ỦY BAN NHÂN DÂN THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC SÀI GÒN**



**TIỂU LUẬN HỌC PHẦN KIỂM THỬ PHẦN MỀM**

SINH VIÊN: PHẠM ĐĂNG HOÀNG

MÃ SỐ SV: 3118410133

LỚP: DCT11810

**TP. HỒ CHÍ MINH, THÁNG 09 NĂM 2021**

**MỤC LỤC**

PHẦN 1. LÝ THUYẾT CƠ SỞ ………………………………………………………….1

PHẦN 2. TÌM HIỂU CÔNG CỤ JUNIT TRONG JAVA ……………………………….4

PHẦN 3. TÌM HIỂU CÔNG CỤ KIỂM THỬ WEB TỰ ĐỘNG SELENIUM ………....7

TÀI LIỆU THAM KHẢO …………………………………………………………………9

**PHẦN 1. LÝ THUYẾT CƠ SỞ**

**Câu 1:**

1. Trình bày khái niệm testcase.

* Test Case là một tập hợp các trường hợp điều kiện theo đó mà Tester có thể dựa vào nó để xác định liệu một ứng dụng, hệ thống phần mềm hoặc là một trong các tính năng của nó có hoạt động như mong muốn cần làm hay không.

1. Trình bày khái niện kiểm thử hộp đen.

* Kiểm thử hộp đen là phương pháp test dựa trên đầu vào và đầu ra của chương trình để test mà không quan tâm tới code bên trong được viết ra sao. Tester xem phần mềm như là một hộp đen. Kiểm thử hộp đen không yêu cầu kỹ sư kiểm thử cần phải có bất kỳ kiếm thức về mã hoặc thuật toán của chương trình. Nó kiểm tra các chức năng của hệ thống tức là nhứng gì hệ thống được cho là cần phải làm dựa trên các đặt tả yêu cầu. Các trường hợp kiểm thử thường được xây dựng xung quanh đó.

**Câu 2:**

1. Hãy lập trình giải bài toán trên sau: Cho 3 số nguyên dương a,b,c có phải là số đo 3 cạnh của một tam giác hay không ? Nếu có hãy kiểm tra xem đó là tam giác đều, vuông, cân hay tam giác thường ?

#include<iostream>

using namespace std;

int main() {

int a,b,c;

cout<<”cạnh a là: “;

cin>>a;

cout<<”cạnh b là: “;

cin>>b;

cout<<”cạnh c là: ”;

cin>>c:

if (a<b+c && b<a+c && c<a+b) {

if (a\*a=b\*b=c\*c || b\*b=a\*a+c\*c || c\*c=a\*a+b\*b)

cout<<”là tam giác vuông”;

else if(a==b && b==c)

cout<<”là tam giác đều”:

else if(a==b || a==c || b==c)

cout<<”là tam giác cân”:

}

else

cout<<”ba cạnh a,b,c không phải là ba cạnh của tam giác”;

return 0;

}

1. Hãy sử dụng một kỹ thuật kiểm thử giá trị biên kiểm thử chương trình trên.

Bài toán chỉ cho biết ba cạnh của tam giác là số nguyên. Nhưng chúng ta biết cận dưới của cạnh tam giác là 1. Và chúng ta giả sử cận trên là 100.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stt | a | b | c | Kết quả |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | 50  50  50  50  50  50  50  50  50  30  50  100  1  2  50  30  99  100 | 50  50  50  40  50  50  1  2  50  50  99  200  50  50  50  40  50  50 | 1  2  50  30  99  100  50  50  50  40  50  100  50  50  50  50  50  50 | Tam giác cân  Tam giác cân  Tam giác đều  Tam giác vuông  Tam giác cân  Không phải là tam giác  Tam giác cân  Tam giác cân  Tam giác đều  Tam giác vuông  Tam giác cân  Không phải là tam giác  Tam giác cân  Tam giác cân  Tam giác đều  Tam giác vuông  Tam giác cân  Không phải tam giác |

**Câu 3:**

1. Trình bày khái niệm kiểm thử hộp trắng.

* Kiểm thử hộp trắng là một phương pháp kiểm thử phần mềm trong đó tester biết về cấu trúc nội bộ / thiết kế. Người kiểm tra chọn đầu vào để thực hiện các đường dẫn thông qua mã và xác định đầu ra thích hợp. Kiến thức lập trình và kiến thức thực hiện là rất cần thiết trong kiểm thử hộp trắng.

1. Trình bày các độ đo kiểm thử cấp 1, cấp 2, cấp 3 cho kiểm thử dòng điều khiển.

* Độ đo kiểm thử cấp 1 (C1): mỗi câu lệnh được thực hiện ít nhất một lần sau khi chạy các ca kiểm thử (test cases).
* Độ đo kiểm thử cấp 2 (C2): các điểm quyết định trong đồ thị dòng điều khiển của đơn vị kiểm thử đều được thực hiện ít nhất một lần cả hai nhánh đúng và sai.
* Độ đo kiểm thử cấp 3 (C3): Với các điều kiện phức tạp (chứa nhiều điều kiện con cơ bản), việc chỉ quan tâm đến giá trị đúng sai là không đủ để kiểm tra tính đúng đắn của chương trình ứng với điều kiện phức tạp này. Với các đơn vị chương trình yêu cầu cao về tính đúng đắn , việc tuân thủ độ đo C3 là hết sức cần thiết. Điều kiện để đảm bảo độ đo này là các điều kiện con thuộc các điều kiện phức tạp tương ứng với các điểm quyết định trong đồ thị dòng điều khiển của đơn vị cần kiểm thử đều được thực hiện ít nhất một lần cả hai nhánh đúng và sai.

**Câu 4:**

**Cho mảng một chiều chứa n số nguyên a0,a1,a2,…,an-1. Hãy viết hàm tìm giá trị lớn nhất trong mảng và thực hiện các công việc sau:**

* Hàm tìm giá trị lớn nhất trong mảng:

1. Int max(int a[], int n) {
2. Int max = a[0];

For (int i=1; i<n (3); i++ (4))

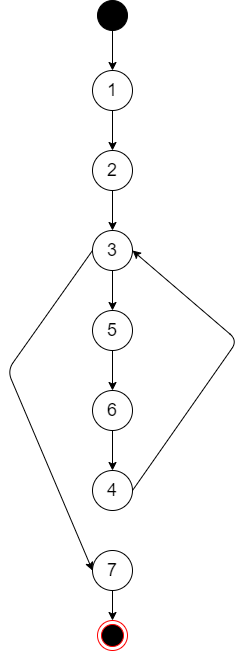
(5) if (max < a[i])

(6) max = a[i];

(7) return max;

}

1. Vẽ đồ thị dòng điều khiển cho hàm trên.

****

1. Xác định đường đi và các ca kiểm thử với độ đo cấp 1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test path | Input | EO | RO | Note |
| TC1 | 1,2,3,5,6,4,3,7 | [4,7], 2 | 7 |  |  |

1. Xác định đường đi và các ca kiểm thử với độ đo cấp 2.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ID | Test path | Input | EO | RO | Note |
| TC1 | 1,2,3,7 | [],0 | 0 |  |  |
| TC2 | 1,2,3,5,6,4,3,7 | [6,9],2 | 9 |  |  |

Với TC1, đường đi tương ứng là 1,2,3,7 thì ta thấy đường đi này không vào vòng lặp đồng nghĩa với việc là điều kiện ở lệnh 3 sai, vì vậy chọn n=0 -> mảng a không có phần tử nào.

**PHẦN 2. TÌM HIỂU CÔNG CỤ JUNIT TRONG JAVA**

**Câu 5A. Xác định các phương thức trong Junit và cho ví dụ từng phương thức. Giới thiệu các chức năng.**

* Các phương thức trong Junit: Trong lớp Junit.Framework.Assert các phương thức dạng assert được dùng để kiểm tra các điều kiện:

+ AssertEquals(): kiểm tra sự bằng nhau của hai giá trị, Test sẽ được thông qua nếu hai giá trị expected và actual bằng nhau.

Ví dụ: Kiểm tra kết quả của một phép nhân có như mong đợi không.

@Test

Public void test Multiplication() {

int number1 = 2;

int number2 = 5;

int expectedVul = 10;

int actualVul = number1 \* number2;

assertEquals(expectedVul, actualVul);

}

Trường hợp này thì test case sẽ được thông qua vì giá trị expectedVul và actualVul là bằng nhau, ở đây chúng ta gán hai giá trị cho hai biến number nhưng sau đó hai giá trị đó sẽ là già trị đầu vào do người dùng nhập.

+ AssertNotEquals() : Kiểm tra sự khác nhau của hai giá trị, Test được thông qua nếu hai giá trị expected và actual khác nhau.

Ví dụ: Kiểm tra giá trị hai số nhân nhau có giống kết quả mong đợi hay không.

@Test

Public void testMultiplication() {

int number1 = 2;

int number2 = 5;

int expectedVul = 10;

int actualVul = number1 \* number2;

assertEquals(expectedVul, actualVul);

}

Cách hoạt động của hàm assertNotEquals() cũng giống như hàm assertEquals(), cũng xét hai biến expectedVul và actualVul nhưng ở đây nếu hai giá trị khác nhau thì mới được thông qua. Trường hợp này thì test case được thông qua vì hai giá trị là khác nhau.

+ assertTrue(): Đánh giá biểu thức luận lý, test được thông qua nếu biểu thức đúng

Ví dụ:

@Test

public void testAssertTrue() {

assertTrue(true);

}

+ assertFalse(): Đánh giá biểu thức luận lý, test được thông qua nếu biểu thức sai.

Ví dụ:

@Test

public void testAssertFalse() {

assertFalse(false);

}

+ AssertArrayEquals(): Kiểm tra sự bằng nhau của hai mảng, Test được thông qua nếu các giá trị của hai mảng là bằng nhau.

Ví dụ:

@Test

public void testArrayEquals() {

int[] array1 = [1,2,3];

int[] array2 = [1,2,3];

assertArrayEquals(array1, array2);

}

+ AssertNull(): So sánh tham chiếu của một đối tượng với giá trị null, test sẽ được thông qua nếu giá trị của tham chiếu đó là null.

Ví dụ:

@Test

public void testNull() {

assertNull(“đây là Null”, null);

}

+ AssertNotNull(): So sánh tham chiếu một đối tượng với giá trị null, test sẽ được thông qua nếu giá trị của tham chiếu đó khác null.

Ví dụ:

@Test

public void testNull() {

assertNull(“đây không phải là Null”, new Object());

}

+ Fail: Phương thức này làm cho test hiện hành thất bại, phương thức này thường được sử dụng khi xử lý trường hợp ngoại lệ.

Ví dụ:

@Test

public void testFail() throws Exception {

Assert.fail(“thất bại”);

}

+ Setup: Là một phương thức nằm trong lớp junit.framework.TestCase phương thức này được thực thi trước các phương thức kiểm thử. Thường phương thức này dùng để khởi tạo những biến chung cần thiết trong quá trình sử dụng để khởi tạo những biến chung cần thiết trong quá trình sử dụng và cũng là phương thức được tận dụng để khởi tạo những điều kiện cần trước khi tiến vào quá trình kiểm thử.

Ví dụ:

@Before

public void setup() {

number1 = 2;

number2 = 4;

}

Hai giá trị number sẽ được khởi tạo trước cho quá trình tính toán về sau.

* Các chức năng của JUnit:

+ JUnit là framework mã nguồn mở, được sử dụng để viết và chạy kiểm thử.

+ Cung cấp annotation để định nghĩa các phương thức kiểm thử (@Before, @Test,…)

+ Cung cấp các Asserttion để kiểm tra kết quả mong đợi.

+ Cung cấp các test runner để thực thi các test script.

+ Test case JUnit có thể được chạy tự động.

+ Test case JUnit có thể được tổ chức thành các test suite.

+ JUnit nhìn kết quả trực quan: pass và fail.

**PHẦN 3. TÌM HIỂU CÔNG CỤ KIỂM THỬ WEB TỰ ĐỘNG SELENIUM**

**Câu 7. Trình bày các chức năng cơ bản.**

* Có thể kết hợp với nhiều ngôn ngữ lập trình: Java, .Net, Ruby, Python, Perl.
* Selenium giả lập thao tác người dùng trên web page và các web element.
* Có thể verify và so sánh thông tin trên web page.
* Script được base theo HTML, học dễ và ứng dụng cũng rất nhanh.
* Có thể tạo một bộ tets bao gồm nhiều test case.
* Có thể run các tets suite thông qua Selenium IDE hoặc Selenium command line.
* Selenium API hỗ trợ nhiều loại trình duyệt, do đó sẽ rất dễ dàng khi test các ứng dụng web với các trình duyệt khác nhau.

**Câu 8. Minh họa từng chức năng.**

* Có thể tạo một bộ test bao gồm nhiều test case:



* Selenium API hỗ trợ nhiều loại trình duyệt, do đó sẽ rất dễ dàng khi test các ứng dụng web và các trình duyệt khác nhau:



**Câu 9. Cho biết sự khác nhau giữa Stress test và Volumn test.**

|  |  |
| --- | --- |
| Stress test | Volumn test |
| Là một loại của Performance Test | Là thử nghiệm của một ứng dụng với số lượng lớn dữ liệu trong cơ sở dữ liệu. |
| Stress test để kiểm tra xem hệ thống hoạt động như thế nào khi quá tải và cách hệ thống phục hồi khi xảy ra lỗi. | Volimn test để xác minh nếu hệ thống đáp ứng một khối lượng dữ liệu nhất định như mong đợi. Nó có thể phải tắng kích thước của file. |
| Mục đích của Stress test nhằm đảm bảo rằng khi hoạt động trong điều kiện tải cao trong một khoảng thời gian cố định sẽ không bị crash. | Volumn test giúp xác định khả năng của hệ thống hoặc ứng dụng – khối lượng bình thường và khối lượng lớn. |
| Stress test kiểm tra thời gian phản hồi ổn định,… | Volumn test kiểm tra việc check nếu mất dữ liệu, thời gian đáp ứng hệ thống, kiểm tra xem dữ liệu lưu trữ chính xác hay không,… |

**Câu 10. Trình bày chức năng thực hiện Stress test và Volumn test.**

* Chức năng thực hiện Stress test:

+ Để kiểm tra xem hệ thống có hoạt động tốt ở các điều kiện bất thường hay không.

+ Hiển thị thông báo lỗi thích hợp kkhi hệ thống xảy ra hoạt động bất thường.

+ Hệ thống xảy ra lỗi khi gặp các điều kiện khắc nghiệt có thể gây ra tổn thất to lớn.

+ Tốt nhất là chuẩn bị trước để tránh các trường hợp xấu nhất có thể xảy ra bằng việc thực hiện stress test.

* Chức năng thực hiện Volumn test:

+ Cách xác định các vấn đề về tải, bạn có thể tiết kiệm rất nhiều tiền cái mà sẽ được chi vào việc bảo trì ứng dụng.

+ Nó giúp trong việc start nhanh hơn các kế hoạch mở rộng khả năng.

+ Xác định sớm các vấn đề trọng yếu.

+ Nó đảm bảo rằng hệ thống của bạn bây giờ có khả năng sử dụng trên thế giới thực.

+ Để tìm ra điểm làm giảm sự ổn định của hệ thống.

**TÀI LIỆU THAM KHẢO**

[1]. <https://viblo.asia/p/ky-thuat-kiem-thu-hop-trang-white-box-testing-maGK7MpOlj2>

[2]. <https://topdev.vn/blog/lam-quen-tom-tat-co-ban-ve-selenium/>

[3]. <https://viblo.asia/p/gioi-thieu-ve-selenium-webdriver-selenium-tutorial-8-6J3ZgONAZmB>