**Thợ lành nghề #5: Bước nhỏ**

Ngày 24 tháng 7 năm 2002.

Jerry yêu cầu tôi viết một chương trình tạo ra các thừa số nguyên tố. Tôi viết xong, chương trình chạy ngon lành và sau đó gã xoá mất chương trình đó. Tôi khá bực nhưng Jerry bảo: “Ðừng có bám rịt mã nguồn của mày quá như vậy.”

Tôi chẳng khoái cái trò này nhưng không có một chút luận cứ nào để chống chọi với gã. Tôi ngồi đó, lặng thinh trong sự bất đồng.

“Được rồi”, gã nói, “Mình làm lại từ đầu. Cách tụi tao làm ở đây là viết kiểm thử đơn vị trước”.

Điều này hiển nhiên là vô lý. Tôi nhanh trí phản ứng ngay: “Hở?”

“Ðể tao chỉ cho mày thấy.” Gã nói. “Nhiệm vụ của tụi mình là tạo ra một mảng các thừa số nguyên tố từ một số nguyên. Mày nghĩ được trường hợp kiểm thử nào đơn giản nhất?”

“Trường hợp hợp lý đầu tiên là 2. Và kết quả cần trả về một mảng với chỉ một số 2.”

“Ðúng rồi.” Gã nói. Và gã viết một kiểm thử đơn vị như sau:

public void testTwo() throws Exception {

int factors[] = PrimeFactorizer.factor(2);

assertEquals(1, factors.length);

assertEquals(2, factors[0]);

}

Tiếp theo, gã viết một đoạn mã rất đơn giản để cho phép cái “test case” ở trên biên dịch.

public class PrimeFactorizer {

public static int[] factor(int multiple) {

return new int[0];

}

}

Gã chạy kiểm thử và nó báo lỗi: **“testTwo(TestPrimeFactors): expected: <1> but was: <0>”**.

Ðến đây gã nhìn tôi và nói: “Hãy làm cách gì đơn giản nhất để vượt qua trường hợp kiểm thử đó.”

Bây giờ thì chồng chất sự vô lý. “Ý ông là sao?” Tôi hỏi. “Ðiều đơn giản nhất hẳn trả về một mảng với số 2 trong đó.”

Gã trả lời với vẻ mặt nghiêm nghị: “Ừ, làm vậy đi.”

“Nhưng ngớ ngẩn quá.” Tôi nói, “Cái mã này sai. Giải pháp thực sự không chỉ trả về có số 2.”

“Ừa, đúng vậy.” Gã đáp lời. “Nhưng chiều lòng tao một chút đi.”

Tôi thở dài bực dọc, đảo mắt nhìn gã, gắt gỏng một chút rồi bắt đầu viết:

public static int[] factor(int multiple) {

return new int[]{2};

}

Tôi chạy cái kiểm thử và tất nhiên nó ổn cả.

Tôi hỏi “Cái này chứng minh được điều gì vậy?”

“Nó chứng minh là mày có thể viết một cái hàm tìm ra thừa số nguyên tố của 2.” Gã nói. “Nó cũng chứng minh là kiểm thử đã ổn khi cái hàm trả về đúng với số 2.”

Tôi đảo mắt lần nữa. Mấy thứ này nằm dưới “cơ” của tôi. Tôi ngỡ làm một tay học việc ở đây sẽ được dạy một cái gì đó cơ chứ.

“Bây giờ, trường hợp kiểm thử nào đơn giản nhất mình có thể đưa thêm vào?” Gã hỏi tôi.

Tôi không kìm được, tôi chì chiết một cách giễu cợt với câu nói: “Ôi, Jerry hay là mình nên thử với số 3?”

Và mặc dù tôi có hy vọng, không ngờ gã viết kiểm thử cho số 3 thật:

public void testThree() throws Exception {

int factors[] = PrimeFactorizer.factor(3);

assertEquals(1, factors.length);

assertEquals(3, factors[0]);

}

Cái kiểm thử này thông báo lỗi như đã đoán trước: **“testThree(TestPrimeFactors): expected: <3> but was: <2>”**.

“Được rồi Alphonse, làm cách gì đơn giản nhất để vượt qua cái kiểm thử này.”

Sốt ruột, tôi vớ lấy bàn phím và gõ vào như sau:

public static int[] factor(int multiple) {

if (multiple == 2) {

return new int[]{2};

} else {

return new int[]{3};

}

}

Tôi chạy mấy cái kiểm thử và chúng đều ổn cả.

Jerry nhìn tôi với một nụ cười bất thường. Gã nói: “Được rồi, mấy cái kiểm thử đó đạt rồi. Tuy nhiên, nhìn mã không sáng sủa phải không?”

Gã là người bày cái trò ngớ ngẩn này và bây giờ gã đi hỏi tôi mã có sáng sủa không? “Tôi nghĩ rằng toàn bộ bài tập này khá lờ mờ đó.” Tôi nói.

Gã lờ đi và tiếp tục. “Cứ mỗi lần mày thêm vào một kiểm thử mới, mày phải làm cho nó đạt bằng cách làm cho mã nguồn tổng quát hơn. Bây giờ thử đưa ra thay đổi đơn giản nhất, tổng quát hơn giải pháp đầu tiên của mày xem sao.”

Tôi nghĩ về vấn đề này chừng một phút. Rốt cuộc Jerry đã hỏi tôi vài điều cần động não. Ðúng vậy, có giải pháp tổng quát hơn nữa. Tôi vớ lấy bàn phím và gõ như sau:

public static int[] factor(int multiple) {

return new int[]{multiple};

}

Các kiểm thử đều ổn cả và Jerry mỉm cười nhưng tôi vẫn không thể hình dung làm sao mấy cái trò này đưa đến giải pháp cho vấn đề tạo ra thừa số nguyên tố. Ðến mức này điều duy nhất tôi có thể phát biểu là những cái trò quái đản này chỉ phí thời gian. Mặc dù vậy, tôi vẫn không ngạc nhiên mấy khi Jerry hỏi tôi: Bây giờ, cái kiểm thử nào đơn giản nhất mình có thể đưa thêm vào?”

“Rõ ràng là cho trường hợp số 4.” Tôi nói một cách thiếu kiên nhẫn rồi vớ lấy bàn phím và viết:

public void testFour() throws Exception {

int factors[] = PrimeFactorizer.factor(4);

assertEquals(2, factors.length);

assertEquals(2, factors[0]);

assertEquals(2, factors[1]);

}

Tôi nói “Tôi nghĩ cái ‘assert’ thứ nhất sẽ hỏng vì sẽ trả về 1 là kích cỡ của mảng.”

Quả vậy, khi chạy kiểm thử cho báo cáo: “**testFour(TestPrimeFactors) :expected <2> but was <1>”**.

Tôi hỏi: “Tôi đoán ông muốn tôi đưa ra thay đổi đơn giản nhất có thể để các kiểm thử đều đạt và tạo ra phương thức thừa số tổng quát hơn?”

Jerry gật đầu.

Tôi cố gắng tập trung giải quyết cho cái kiểm thử trước mắt, lờ các kiểm thử tôi biết sẽ đụng đến sau. Cái trò này thật ai oán nhưng Jerry muốn vậy. Kết quả như sau:

public class PrimeFactorizer {

public static int[] factor(int multiple) {

int currentFactor = 0;

int factorRegister[] = new int[2];

for (; (multiple % 2) == 0; multiple /= 2) {

factorRegister[currentFactor++] = 2;

}

if (multiple != 1) {

factorRegister[currentFactor++] = multiple;

}

int factors[] = new int[currentFactor];

for (int i = 0; i &lt; currentFactor; i++) {

factors[i] = factorRegister[i];

}

return factors;

}

}

Ðoạn mã này vượt qua tất cả các kiểm thử, nhưng nhìn khá lộn xộn. Jerry nhăn mặt như thể gã đánh được mùi hôi thối đâu đây. Gã nói: “Mình phải tái cấu trúc cái này trước khi đi tiếp.”

“Đợi đã.” Tôi phản đối. “Tôi đồng ý nó lộn xộn nhưng sao mình không làm cho nó chạy trước rồi tái cấu trúc lại nếu có đủ thời gian?”

“Trời! Không được!” Jerry nói. “Mình cần phải tái cấu trúc ngay lúc này để có thể thấy cấu trúc thực sự tiến hoá, không thì chúng ta chỉ chồng chất cái bừa bộn trên cái bừa bộn và chúng ta sẽ không còn biết mình đang làm gì nữa.”

“Được thôi.” Tôi thở dài. “Thì dọn dẹp.”

Thế rồi bọn tôi tiến hành tái cấu trúc một chút. Kết quả như sau:

public class PrimeFactorizer {

private static int factorIndex;

private static int[] factorRegister;

public static int[] factor(int multiple) {

initialize();

findPrimeFactors(multiple);

return copyToResult();

}

private static void initialize() {

factorIndex = 0;

factorRegister = new int[2];

}

private static void findPrimeFactors(int multiple) {

for (; (multiple % 2) == 0; multiple /= 2) {

factorRegister[factorIndex++] = 2;

}

if (multiple != 1) {

factorRegister[factorIndex++] = multiple;

}

}

private static int[] copyToResult() {

int factors[] = new int[factorIndex];

for (int i = 0; i &lt; factorIndex; i++) {

factors[i] = factorRegister[i];

}

return factors;

}

}

Jerry tuyên bố: “Ðến lúc cho cái kiểm thử tiếp theo.” và gã chuyển bàn phím cho tôi.

Tôi vẫn chưa thể nhận ra trò này đi đến đâu nhưng biết rằng không có cách nào để thoát ra được. Một cách nhân nhượng tôi viết cái kiểm thử như sau:

public void testFive() throws Exception {

int factors[] = PrimeFactorizer.factor(5);

assertEquals(1, factors.length);

assertEquals(5, factors[0]);

}

“Thật là lý thú.” Tôi nói trong khi nhìn chằm chặp vào cái thanh màu xanh (thanh trạng thái kiểm thử, màu xanh tức là đạt), “nó chạy mà chẳng cần thay đổi gì hết.”

“Ðúng là lý thú”. Jerry nối tiếp. “Hãy thử với kiểm thử tiếp theo.”

Lúc này tôi rõ ràng đã bị thu hút. Tôi không kỳ vọng các trường hợp kiểm thử làm việc như vậy. Mặc dù nghĩ về vấn đề này, tôi vẫn chưa thực đoán được lý do vì sao nó vẫn chạy tốt. Tôi khá chắc việc trường hợp kiểm thử tiếp theo sẽ hỏng nên đã viết như sau và chạy thử:

public void testSix() throws Exception {

int factors[] = PrimeFactorizer.factor(6);

assertEquals(2, factors.length);

assertContains(factors, 2);

assertContains(factors, 3);

}

private void assertContains(int factors[], int n) {

String error = &quot;assertContains:&quot; + n;

for (int i = 0; i &lt; factors.length; i++) {

if (factors[i] == n) {

return;

}

}

fail(error);

}

“Úi! Cái kiểm thử này cũng ổn luôn!” tôi rú lên.

“Lý thú.” Jerry gật gù. “Vậy 7 sẽ chạy luôn phải không?”

“Vâng, tôi nghĩ vậy.”

“Vậy thì bỏ nó đi và đi thẳng tới 8, nó sẽ không qua được trường hợp này đâu!”

Gã đúng. Với trường hợp của 8 sẽ hỏng vì mảng factorRegister quá nhỏ.

public void testEight() throws Exception {

int factors[] = PrimeFactorizer.factor(8);

assertEquals(3, factors.length);

assertContainsMany(factors, 3, 2);

}

private void assertContainsMany(int factors[], int n, int f) {

String error = &quot;assertContains(&quot; + n + &quot;,&quot; + f + &quot;)&quot;;

int count = 0;

for (int i = 0; i &lt; factors.length; i++) {

if (factors[i] == f) {

count++;

}

}

if (count != n) {

fail(error);

}

}

“Ðúng là nhẹ nhõm! nó hỏng rồi!”

“Ừa.” Jerry đáp “Vì vượt quá kích thước của mảng. Mày có thể làm nó vượt qua được bằng cách gia tăng kích thước của factorRegister nhưng cách này không tổng quát hơn được. Thì cứ thử xem sao rồi mình giải quyết vấn đề chiều dài của mảng sau.”

Thế là tôi đổi 2 thành 3 trong hàm initialize và tôi có cái thanh màu xanh.

“Được rồi,” tôi nói. “tối đa các thừa số mà một số có thể có là bao nhiêu?”

“Tao nghĩ là logarit cơ số 2 của số đó thì phải.” Jerry nói.

“Đợi đã!” Tôi nói, “Có thể mình đang đi lòng vòng đấy. Số lớn nhất mình có thể xử lý là mấy? không phải là 2 mũ 64 sao?”

Jerry đáp “Tao chắc là không thể lớn hơn con số đó.”

“Được rồi, vậy thì thử tạo ra chiều dài của factorRegister là 100 đi. Nó lớn đủ để xử lý bất cứ số nào mình quẳng cho nó.”

“Ðược thôi.” Jerry nói “100 số nguyên thì chẳng có gì phải lo.”

Chúng tôi thử điều này và các kiểm thử vẫn chạy.

Tôi nhìn Jerry và nói: “kiểm thử tiếp theo của tôi là 9. Chắc chắn nó sẽ hỏng.”

Gã đáp “Thì thử đi.”

Vậy là tôi viết mã như sau:

public void testNine() throws Exception {

int factors[] = PrimeFactorizer.factor(9);

assertEquals(2, factors.length);

assertContainsMany(factors, 2, 3);

}

“Trời, nó hỏng thật.” Tôi nói. “Vượt qua trường hợp này cũng đơn giản thôi. Tôi chỉ cần bỏ đi 2 như một số đặc biệt trong findPrimeFactors và dùng cả 2 và 3 cho thuật toán tổng quát.” Thế là tôi điều chỉnh hàm findPrimeFactors như sau:

private static void findPrimeFactors(int multiple) {

for (int factor = 2; multiple != 1; factor++) {

for (; (multiple % factor) == 0; multiple /= factor) {

factorRegister[factorIndex++] = factor;

}

}

}

“Được rồi, nó đã chạy”. Jerry nói. “Bây giờ xem thử cái kiểm thử tiếp theo nào sẽ hỏng?”

“Ừm, thuật toán đơn giản tôi dùng để chia được từ số phi nguyên tố lẫn số nguyên tố. Kiểu này sẽ không thực hiện cho đúng được nên phiên bản đầu của chương trình chỉ chia được từ số nguyên tố. Thuật toán đầu dành cho số phi nguyên tố sẽ chia cho 4 nên tôi mường tượng 4X4 sẽ hỏng.

public void testSixteen() throws Exception {

int factors[] = PrimeFactorizer.factor(16);

assertEquals(4, factors.length);

assertContainsMany(factors, 4, 2);

}

“Ui! Cái kiểm thử này chạy rồi.” Tôi nói. “Làm sao nó qua được nhỉ?”

“Nó qua được vì tất cả các số 2 đã được loại bỏ trước khi mày thử chia cho 4, nên 4 không bao giờ nhận ra như một thừa số. Nên nhớ, nó cũng không thấy như một thừa số với 8, hoặc là 4!”

“Tất nhiên!” tôi trả lời. “Tất cả các số nguyên tố bị dời bỏ trước các đa hợp. Thật ra thuật toán dùng để kiểm tra các đa hợp không liên quan gì hết, nhưng điều đó có nghĩa là tôi không hề cần dãy của các số nguyên tố trong phiên bản ban đầu của mình.”

“Ðúng thế.” Jerry nói. “Ðó là lý do tại sao tao xoá nó.”

“Vậy thì xong? Mình hoàn thành rồi phải không?”

Jerry hỏi: ” Mày có thể nghĩ ra được cái test case nào bị hỏng không?”

“Tôi không biết nữa, hãy thử 1000 đi.” Tôi trả lời.

“À, kiểu chơi ôm đồm. Được rồi, thử đi.”

public void testThousand() throws Exception {

int factors[] = PrimeFactorizer.factor(1000);

assertEquals(6, factors.length);

assertContainsMany(factors, 3, 2);

assertContainsMany(factors, 3, 5);

}

“Nó chạy luôn! Được rồi, hay là…”

Chúng tôi viết nhiều kiểm thử khác nhưng cái nào cũng ổn cả. Phiên bản này của chương trình đơn giản hơn phiên bản đầu tiên của tôi nhiều và chạy nhanh hơn nữa. Hèn chi Jerry đã xoá đi phiên bản đầu.

Ðiều làm tôi kinh ngạc và vẫn còn làm tôi kinh ngạc là sau mỗi kiểm thử chúng tôi lại cho ra một giải pháp tốt hơn. Nếu không tiến lên với mỗi kiểm thử thì tôi không nghĩ sẽ có thể triển khai theo cách đơn giản này. Tôi không biết chuyện gì sẽ xảy ra với những dự án lớn hơn nữa?

Hôm nay tôi đã học được đôi điều.

**Bài tiếp:** [Thợ lành nghề #6: Một lần không đủ (Dịch vụ Socket 1)](https://codegym.vn/blog/tho-lanh-nghe-6-mot-lan-khong-du-dich-vu-socket-1/)

**Bài trước:** [Thợ lành nghề #4 – Bài kiểm tra tính kiên nhẫn](https://codegym.vn/blog/tho-lanh-nghe-4-bai-kiem-tra-tinh-kien-nhan/)

**Tác giả:** Robert C. Martin

**Người dịch:** Hoàng Ngọc Diêu (conmale)