**Thợ lành nghề #3 - Tính rõ ràng và sự cộng tác**

*Ngày 13 tháng 2 năm 2002.*

*Lần trước, Jerry, một cựu học việc yêu cầu tay học việc Alphonse viết một chương trình tạo số nguyên tố dùng phương pháp Sàng của Eratosthenes. Jerry xem xét và giúp Alphonse* [*tái cấu trúc mã nguồn*](http://tapchilaptrinh.wordpress.com/2012/10/05/tai-cau-truc-ma-nguon/) *đó. Anh ta không hài lòng với công việc của Alphonse. Lần trước Alphonse thực hiện xong một phần việc* [*tái cấu trúc mã nguồn*](http://tapchilaptrinh.wordpress.com/2012/10/05/tai-cau-truc-ma-nguon/) *và nghĩ chắc Jerry sẽ chấp thuận…*

…Jerry chỉ thoáng gật đầu. Liệu gã có thật sự thích những điều tôi đã làm không?

Sau đó Jerry đọc toàn bộ chương trình, từ đầu đến cuối như thể đang đọc bài chứng minh hình học. Gã bảo tôi đây là một bước hết sức quan trọng. “Ðến bước này, tụi mình đã thực hiện [tái cấu trúc](http://tapchilaptrinh.wordpress.com/2012/10/05/tai-cau-truc-ma-nguon/) các phân mảnh của chương trình. Bây giờ tụi mình xem liệu toàn bộ chương trình có gắn kết với nhau như một tổng thể đọc được.”

Tôi hỏi: “Jerry, anh cũng làm như thế với chính mã nguồn của mình sao?”

Jerry quắc mắt lên và nói: “Ở đây tụi tao làm việc với nhau theo nhóm nên không có cái mã nào là của riêng tao hết. Bộ mày cho rằng cái mã này của riêng mày hở?”

Tôi trả lời hết sức nhỏ nhẹ: “Hết nghĩ như vậy rồi, anh có ảnh hưởng rất lớn đến mã nguồn này.”

Gã trả lời: “Cả hai thằng mình đều ảnh hưởng đến nó, và đây là cách ông C chuộng. Ông ấy không muốn riêng một ai làm chủ mã nguồn hết đâu. Câu trả lời cho câu hỏi của mày: “Ðúng vậy, ở đây tụi tao thực hành tái cấu trúc và dọn rác. Đó là cách làm của ông C.”

Trong khi đọc mã nguồn, Jerry nhận thấy gã không thích cái tên **initializeArrayOfIntegers**.

Gã nói: “Cái được khởi tạo ở đây thực ra không phải là một mảng số nguyên, mà là một mảng boolean. Nhưng **initializeArrayOfBooleans** không phải là một cải tiến. Ðiều chúng ta thực sự muốn làm ở hàm này là liệt kê ra một danh sách các số nguyên phù hợp và để chúng lên một cái sàng, rồi sau đó lọc và loại ra các số không phải số nguyên tố (tức là loại ra những bội số)”. (Do đó, danh sách lúc đầu sẽ không bị gạch chéo, những số bị loại sẽ sẽ bị gạch chéo (crossed out)).

Tôi trả lời: “Tất nhiên rồi!” Thế là tôi vớ lấy bàn phím và sửa tên của hàm đó thành **uncrossIntegersUpTo**. Tôi cũng thấy không khoái cái tên **isCrossed** lại dùng cho một mảng boolean, nên tôi đổi nó thành **crossedOut**. Các kiểm thử vẫn chạy. Tôi bắt đầu thấy thích mấy cái trò này nhưng Jerry vẫn chẳng hề tỏ vẻ đồng tình.

Sau đó Jerry quay lại và hỏi tôi có phải tôi đã mơ màng theo khói thuốc khi viết cái mớ **maxPrimeFactor**. (Xem **Mã dẫn 6**). Thoạt đầu tôi hết sức ngỡ ngàng nhưng khi xem lại đoạn mã và các chú thích tôi nhận thấy gã có lý. Eo ôi, tôi thấy mình thật là ngu! Căn bậc 2 của chiều dài một mảng số không hẳn là số nguyên. Hàm đó không tính thừa số nguyên tố lớn nhất. Phần chú giải sai bét, hết sức ngượng ngùng tôi viết lại phần chú thích để giải thích rõ hơn cái căn bậc 2 này dùng để làm gì và đổi tên biến, hàm cho thích hợp. Các kiểm thử vẫn chạy.

**Mã dẫn 6.** TestGeneratePrimes.java (một phần)

public class TestGeneratePrimes {

private static int calcMaxPrimeFactor() {

// We cross out all multiples of p, where p is prime.

// Thus, all crossed out multiples have p and q for factors.

// If p &gt; sqrt of the size of the array, then q will never

// be greater than 1. Thus p is the largest prime factor

// in the array, and is also the iteration limit.

double maxPrimeFactor = Math.sqrt(isCrossed.length) + 1;

return (int) maxPrimeFactor;

}

}

“dùng +1 ở đây làm quái gì vậy?” Jerry tru tréo lên.

Tôi nuốt ực một cái, xem lại đoạn mã và cuối cùng tôi phát biểu: “Tôi ngại là khi chỉ lấy phần nguyên của căn bậc 2, thì phần thập phân của căn bậc 2 đó bị mất đi, do đó vòng lặp có thể bị thiếu.”

Gã bèn hỏi: “Cho nên mày xả rác trong đoạn mã với phần “+1” bởi vì mày bị hoảng? Như thế thì ngốc quá, dẹp ngay cái trò gia tăng “+1″ đó và chạy kiểm thử lại đi.”

Tôi làm như thế và toàn bộ các kiểm thử đều vẫn chạy tốt. Tôi nghĩ lại phần này một lúc vì nó làm tôi run quá. Thế nhưng tôi quyết định có thể giới hạn lặp lại thực sự chính là số “thừa số nguyên tố lớn nhất” và “thừa số nguyên tố” đó nhỏ hơn hoặc bằng căn bậc 2 chiều dài của mảng.

“Phần thay đổi vừa rồi làm tôi khá bối rối”. Tôi nói với Jerry. “Tôi hiểu nguồn gốc đằng sau cái căn bậc 2, nhưng tôi cảm thấy không yên, biết đâu có trường hợp “biên” nào đó mình chưa thấy hết.”

Gã lầm bầm “OK, vậy thì viết một cái kiểm thử khác để kiểm tra chuyện đó đi.”

“Tôi nghĩ tôi có thể kiểm tra xem trong các danh sách số nguyên từ 2 đến 500 không có trường hợp ở trên”.

“OK, nếu nó làm cho mày cảm thấy dễ chịu hơn, thì thử đi.” Gã nói. Rõ ràng là gã bắt đầu trở nên mất kiên nhẫn.

Thế là tôi viết hàm **testExhaustive** như trong **Mã dẫn 8**. Phần kiểm thử mới này chạy đúng và nỗi lo sợ của tôi lắng xuống.

Jerry dịu xuống một chút. Gã nói: “Biết được lý do tại sao một cái gì đó chạy được luôn luôn là một điều tốt; và lại càng tốt hơn khi mà kiểm chứng được mày đúng bằng kiểm thử.”

Sau đó Jerry dò qua trọn bộ mã nguồn và các cái kiểm thử một lần nữa (xem **Mã dẫn 7 và 8**). Gã ngã người ra và suy nghĩ chừng một phút rồi nói: “Được rồi, tao nghĩ là tụi mình đã làm xong. Mã nguồn này xem ra đủ rõ ràng (clean) rồi đó. Tao sẽ đưa cho ông C xem.”

Thế rồi gã nhìn tôi, lạnh lùng nói: “Phải nhớ, từ nay về sau khi mày viết một phần nào đó, nên tìm sự giúp đỡ và nhớ giữ cho mã nguồn rõ ràng. Nếu mày nhúng tay vào những thứ dưới tiêu chuẩn này, thì không “thọ” được ở đây đâu.”

Gã rảo bước.

**Mã dẫn 7.** PrimeGenerator.java (cuối cùng)

/\*\*

\* This class generates prime numbers up to a user specified maximum. The

\* algorithm used is the Sieve of Eratosthenes. Given an array of \* integers

\* starting at 2: Find the first uncrossed integer, and cross out all its

\* multiples. Repeat until there are no more multiples in the array.

\*/

public class PrimeGenerator {

private static boolean[] crossedOut;

private static int[] result;

public static int[] generatePrimes(int maxValue) {

if (maxValue &lt; 2) {

return new int[0];

} else {

uncrossIntegersUpTo(maxValue);

crossOutMultiples();

putUncrossedIntegersIntoResult();

return result;

}

}

private static void uncrossIntegersUpTo(int maxValue) {

crossedOut = new boolean[maxValue + 1];

for (int i = 2; i &lt; crossedOut.length; i++) {

crossedOut[i] = false;

}

}

private static void crossOutMultiples() {

int limit = determineIterationLimit();

for (int i = 2; i &lt;= limit; i++) {

if (notCrossed(i)) {

crossOutMultiplesOf(i);

}

}

}

private static int determineIterationLimit() {

// Every multiple in the array has a prime factor that is

// less than or equal to the sqrt of the array size, so we

// don't have to cross out multiples of numbers larger than that root.

double iterationLimit = Math.sqrt(crossedOut.length);

return (int) iterationLimit;

}

private static void crossOutMultiplesOf(int i) {

for (int multiple = 2 \* i; multiple &lt; crossedOut.length; multiple += i) {

crossedOut[multiple] = true;

}

}

private static boolean notCrossed(int i) {

return crossedOut[i] == false;

}

private static void putUncrossedIntegersIntoResult() {

result = new int[numberOfUncrossedIntegers()];

for (int j = 0, i = 2; i &lt; crossedOut.length; i++) {

if (notCrossed(i)) {

result[j++] = i;

}

}

}

private static int numberOfUncrossedIntegers() {

int count = 0;

for (int i = 2; i &lt; crossedOut.length; i++) {

if (notCrossed(i)) {

count++;

}

}

return count;

}

}

**Mã dẫn 8.** TestGeneratePrimes.java (cuối cùng)

import junit.framework.\*;

public class TestGeneratePrimes extends TestCase {

public static void main(String args[]) {

junit.swingui.TestRunner.main(

new String[]{&quot;TestGeneratePrimes&quot;});

}

public TestGeneratePrimes(String name) {

super(name);

}

public void testPrimes() {

int[] nullArray = PrimeGenerator.generatePrimes(0);

assertEquals(nullArray.length, 0);

int[] minArray = PrimeGenerator.generatePrimes(2);

assertEquals(minArray.length, 1);

assertEquals(minArray[0], 2);

int[] threeArray = PrimeGenerator.generatePrimes(3);

assertEquals(threeArray.length, 2);

assertEquals(threeArray[0], 2);

assertEquals(threeArray[1], 3);

int[] centArray = PrimeGenerator.generatePrimes(100);

assertEquals(centArray.length, 25);

assertEquals(centArray[24], 97);

}

public void testExhaustive() {

for (int i = 2; i &lt; 500; i++) {

verifyPrimeList(PrimeGenerator.generatePrimes(i));

}

}

private void verifyPrimeList(int[] list) {

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

verifyPrime(list[i]);

}

}

private void verifyPrime(int n) {

for (int factor = 2; factor &lt; n; factor++) {

assert (n % factor != 0);

}

}

}

Quả là tai hoạ! Tôi cứ ngỡ là giải pháp nguyên thủy của tôi là thượng hạng. Chút gì đó tôi vẫn còn cảm thấy như vậy. Tôi cố phô trương tài năng của mình nhưng tôi đoán là ông C đánh giá cao sự cộng tác và tính minh bạch hơn tài năng cá nhân.

Tôi phải thú nhận rằng chương trình này dễ xem hơn lúc khởi đầu. Nó lại làm việc ngon hơn một tí nữa. Tôi khá hài lòng với kết quả và, mặc dù Jerry có thái độ cộc cằn, nhưng làm việc với gã tôi cũng thấy vui. Tôi học hỏi được rất nhiều.

Dẫu vậy, tôi thấy hơi chùn bước với chính hiệu suất của mình. Tôi không dám nghĩ là mấy tay ở đây sẽ khoái tôi cho lắm. Tôi cũng không dám chắc đến bao bao giờ họ đánh giá tôi đủ “ngon”. Sự thể sẽ khó khăn hơn tôi nghĩ nhiều lắm.

**Bài tiếp:** [Thợ lành nghề #4 – Bài kiểm tra tính kiên nhẫn](https://codegym.vn/blog/2018/08/02/tho-lanh-nghe-4-bai-kiem-tra-tinh-kien-nhan/)

**Bài trước:** [Thợ lành nghề #2 – Chế độ ăn kiêng tăng cường](https://codegym.vn/blog/2018/07/31/tho-lanh-nghe-2-che-do-an-kieng-tang-cuong/)

**Tác giả:** Robert C. Martin

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