**Thợ lành nghề #2 - Chế độ ăn kiêng tăng cường**

Ngày 13 tháng 2 năm 2002 Nhật ký thân mến, Hôm nay đúng là một ngày xui xẻo – Tôi làm hỏng mọi chuyện. Tôi rất muốn gây ấn tượng với các ngài “cựu học việc” ở đây nhưng rút cuộc chỉ làm rối tung cả lên. Ðó là ngày đầu tiên tôi được làm một chân học việc của ông C. Tôi quả rất may mắn mới có được vị trí này. Ông C là một người có tiếng trong giới phát triển phần mềm. Cuộc đua để giành được chân học việc này đúng là nảy lửa. Những người học việc của ông C thường trở nên sáng giá. Ðiều này có nghĩa được làm việc với ông C rõ ràng đem lại giá trị.

*Trong phần trước Jerry, một cựu học việc yêu cầu Alphonse, một người học việc, viết một chương trình tạo các số nguyên dùng* [*Sàng Eratosthenes*](http://vi.wikipedia.org/wiki/S%C3%A0ng_Eratosthenes)*. Jerry nhận thấy Alphonse cài đặt toàn bộ thuật toán vào một hàm khổng lồ, nên đã yêu cầu Alphonse tách nó ra theo ba thành phần: khởi động, ứng tạo và chuẩn bị đầu ra;… nhưng Alphonse không biết phải bắt đầu từ đâu…*

Gã nhìn tôi một lúc, rõ ràng đang đợi tôi làm gì đó. Nhưng rốt cuộc gã thở dài, lắc đầu và tiếp tục. “Ðể diễn tả rõ hơn ba khái niệm này, tao muốn mày tách chúng ra thành ba hàm riêng biệt. Ðồng thời bỏ hết những chú thích không cần thiết và tìm một cái tên khá hơn cho lớp. Khi làm xong những thứ đó, mày phải bảo đảm là những cái kiểm thử vẫn chạy được.” Các bạn có thể thấy những gì tôi đã làm trong Mã dẫn 3. Tôi đánh dấu những thay đổi bằng cách chú thích dòng thay đổi, tương tự như Martin Fowler trình bày trong cuốn Tái cấu trúc ([Refactoring](http://tapchilaptrinh.wordpress.com/2012/10/05/tai-cau-truc-ma-nguon/)). Tôi đổi tên lớp thành một danh từ, bỏ hết những chú thích về Eratosthenes và tạo ra ba hàm tương ứng với ba khái niệm trong hàm *generatePrimes*. Việc tách thành ba hàm buộc tôi phải đưa một số biến cục bộ của hàm thành thuộc tính của lớp. Jerry nói cách này thể hiện rõ hơn những biến nào là cục bộ và biến hàm nào có ảnh hưởng rộng hơn. **Mã dẫn 3.** *PrimeGenerator.java, phiên bản 2*

/\*\*

\* 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 the first uncrossed integer exceeds the square root

\* of the maximum value.

\*/

import java.util.\*;

public class PrimeGenerator {

private static int s;

private static boolean[] f;

private static int[] primes;

public static int[] generatePrimes(int maxValue) {

if (maxValue < 2) {

return new int[0];

} else {

initializeSieve(maxValue);

sieve();

loadPrimes();

return primes; // return the primes

}

}

private static void loadPrimes() {

int i;

int j;

// how many primes are there?

int count = 0;

for (i = 0; i < s; i++) {

if (f[i]) {

count++; // bump count.

}

}

primes = new int[count];

// move the primes into the result for (i = 0, j = 0; i < s; i++)

{

if (f[i]) // if prime

{

primes[j++] = i;

}

}

}

private static void sieve() {

int i;

int j;

for (i = 2; i < Math.sqrt(s) + 1; i++) {

if (f[i]) // if i is uncrossed, cross out its multiples.

{

for (j = 2 \* i; j < s; j += i) {

f[j] = false; // multiple is not prime

}

}

}

}

private static void initializeSieve(int maxValue) {

// declarations

s = maxValue + 1; // size of array

f = new boolean[s];

int i;

// initialize array to true.

for (i = 0; i < s; i++) {

f[i] = true;

}

// get rid of known non-primes

f[0] = f[1] = false;

}

}

Jerry bảo tôi mã nguồn này hơi lộn xộn, nên gã giành lấy bàn phím và chỉ cho tôi cách dọn dẹp. Mã dẫn 4 minh hoạ những gì gã đã làm. Đầu tiên gã vứt đi biến cục bộ *s* trong *initializeSieve* và thay thế nó bằng *f.length*. Sau đó gã đổi tên của ba hàm cho rõ nghĩa hơn. Cuối cùng gã sắp xếp lại cái “bộ lòng”*initializeArrayOfIntegers (*từ *initializeSieve)* để cho dễ đọc hơn một chút. Các cái kiểm thử vẫn chạy tốt. **Mã dẫn 4.** PrimeGenerator.java, phiên bản 3 (một phần)

public class PrimeGenerator {

private static boolean[] f;

private static int[] result;

public static int[] generatePrimes(int maxValue) {

if (maxValue < 2) {

return new int[0];

} else {

initializeArrayOfIntegers(maxValue);

crossOutMultiples();

putUncrossedIntegersIntoResult();

return result;

}

}

private static void initializeArrayOfIntegers(int maxValue) {

f = new boolean[maxValue + 1];

f[0] = f[1] = false; //neither primes nor multiples.

for (int i = 2; i < f.length; i++) {

f[i] = true;

}

}

Tôi phải công nhận là mã nguồn này rõ ràng hơn một chút. Trước giờ tôi nghĩ việc đặt tên dài có tính mô tả cho hàm là phung phí thời giờ, nhưng những điều chỉnh của gã quả thật làm cho mã nguồn dễ đọc hơn. Tiếp theo Jerry trỏ vào *crossOutMultiples và* nói là gã nghĩ có thể làm những lệnh *if(f[i] == true)* dễ đọc hơn nữa. Tôi nghĩ về điểm này chừng một phút. Ý định của những lệnh này là kiểm tra xem liệu *i có chưa bị loại*; thế là tôi đổi tên của *f* thành *unCrossed.* Jerry nói mã này được hơn nhưng gã vẫn chưa hài lòng vì nó dẫn đến khả năng phủ định kép như *unCrossed[i] = false*. Bởi thế gã đổi tên mảng thành *isCrossed* và đổi ý nghĩa của mọi giá trị luận lý. Sau đó gã chạy mọi kiểm thử. Jerry xóa mã gán *true* cho *isCrossed*[0] và *isCrossed*[1]*.* Gã nói điều này đủ tốt để đảm bảo không có phần nào của hàm dùng mảng *isCrossed* với chỉ số nhỏ hơn 2. Mọi kiểm thử vẫn chạy. Jerry tách phần lặp bên trong của hàm *crossOutMultiples* ravà đặt tên là *crossOutMultipleOf*. Gã bảo rằng các lệnh tương tự như *if (isCrossed[i] == false)* dễ gây nhầm lẫn, nên gã tạo hàm có tên *notCrossed* và thay cụm *if* thành *if (notCrossed(i))*. Kế tiếp gã chạy lại mấy kiểm thử. Sau đó Jerry hỏi tôi ý nghĩa của căn bậc hai tôi đã dùng. Tôi tốn ít thời giờ viết chú thích cho lý do cần phải lặp lại cho đến căn bậc hai của độ dài mảng. Tôi cố làm theo Jerry bằng cách tách phần tính toán thành một hàm, tôi có thể viết chú thích trong hàm này. Trong khi viết chú thích tôi nhận ra rằng căn bậc hai là thừa số nguyên tố lớn nhất của bất kỳ số nào trong mảng. Sau đó tôi chọn tên cho các hàm và biến. Cuối cùng tôi bảo đảm các kiểm thử vẫn chạy. Kết quả của các thay đổi trong **Mã dẫn 5**. **Mã dẫn 5.** PrimeGenerator.java phiên bản 4 (một phần)

public class PrimeGenerator {

private static boolean[] isCrossed;

private static int[] result;

public static int[] generatePrimes(int maxValue) {

if (maxValue < 2) {

return new int[0];

} else {

initializeArrayOfIntegers(maxValue);

crossOutMultiples();

putUncrossedIntegersIntoResult();

return result;

}

}

private static void initializeArrayOfIntegers(int maxValue) {

isCrossed = new boolean[maxValue + 1];

for (int i = 2; i < isCrossed.length; i++) {

isCrossed[i] = false;

}

}

private static void crossOutMultiples() {

int maxPrimeFactor = calcMaxPrimeFactor();

for (int i = 2; i <= maxPrimeFactor; i++) {

if (notCrossed(i)) {

crossOutMultiplesOf(i);

}

}

}

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 > 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;

}

private static void crossOutMultiplesOf(int i) {

for (int multiple = 2 \* i;

multiple < isCrossed.length;

multiple += i) {

isCrossed[multiple] = true;

}

}

private static boolean notCrossed(int i) {

return isCrossed[i] == false;

}

Tôi bắt đầu nắm bắt được vấn đề nên liền xét hàm *putUncrossedIntegersIntoResult*. Tôi thấy hàm này có hai phần. Phần thứ nhất đếm các số nguyên không bị loại trong mảng, và tạo nên mảng kết quả (bằng chiều dài của mảng). Phần thứ hai dời các số nguyên không bị loại vào dãy kết quả này. Bởi thế, như bạn thấy trong Mã dẫn6, tôi tách phần thứ nhất ra để hình thành hàm cho chính nó và dọn dẹp đôi chút lặt vặt. Các kiểm thử vẫn chạy. Jerry chỉ thoáng gật đầu. Gã có thật sự khoái những điều tôi đã thực hiện không? **Mã dẫn 6.** PrimeGenerator.java, phiên bản 5 (một phần)

private static void putUncrossedIntegersIntoResult() {

result = new int[numberOfUncrossedIntegers()];

for (int j = 0, i = 2; i < isCrossed.length; i++) {

if (notCrossed(i)) {

result[j++] = i;

}

}

}

private static int numberOfUncrossedIntegers() {

int count = 0;

for (int i = 2; i < isCrossed.length; i++) {

if (notCrossed(i)) {

count++;

}

return count;

}

}

**Bài tiếp:** [Thợ lành nghề #3 – Tính rõ ràng và sự cộng tác](https://codegym.vn/blog/2018/08/01/tho-lanh-nghe-3-tinh-ro-rang-va-su-cong-tac/)

**Bài trước:** [Thợ lành nghề #1 – Mở đầu Thảm họa](https://codegym.vn/blog/2018/07/30/tho-lanh-nghe-1-mo-dau-tham-hoa/)

**Tác giả:** Robert C. Martin

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