CODEGYM

Raising the bar

HỆ THỐNG ĐÀO TẠO LẬP TRÌNH HIỆN ĐẠI

Sách **HỌC TIẾNG ANH SONG NGỮ** cho **LẬP TRÌNH VIÊN** *(Phần 1)*

Phiên bản 1.0 - Tháng 02/2022

LỜI MỞ ĐẦU

Lập trình viên muốn giỏi tiếng Anh "như giỏi code", ngoài những phương pháp học đơn thuần như xem phim, học qua bài hát, video thì việc học qua những bài mẫu song ngữ và đã có highlight cụm từ quan trọng cũng vô cùng thú vị và hiệu quả.

"Sách Song Ngữ Cho Lập Trình Viên (Phần 1)" được biên tập theo chương "Principles, Patterns, and Practices" từ cuốn sách "Agile Software Development" của tác giả Robert C. Martin. Nội dung kể về nhân vật Alphonse, một sinh viên IT mới ra trường và được nhận vào thực tập ở công ty của ông C (chính tác giả).

Thông qua những câu chuyện, tình huống, bài học khi mới đặt chân vào ngành công nghiệp phần mềm được kể lại từ dưới chính góc nhìn Alphonse, các bạn có thể:
- Được tiếp cận với ngôn ngữ chuẩn hơn
- Được mở mang hiểu biết thông qua nguồn kiến thức, ý tưởng của tác giả từ các bài mẫu Anh - Việt
- Vừa được tiếp xúc với tiếng Anh, vừa được ôn lại những bài toán lập trình cơ bản

Mặc dù đội ngũ biên tập của CodeGym đã nỗ lực trong việc hoàn thiện cuốn sách này với tiêu chí dễ hiểu, khoa học và hiệu quả, tuy nhiên khó để tránh khỏi các sai sót trong quá trình biên soạn. Vì vậy, chúng mình rất mong nhận được các ý kiến phản hồi và đóng góp của mọi người thông qua email marketing@codegym.vn.

Cảm ơn các bạn đã tải và đọc "Sách Song Ngữ Cho Lập Trình Viên (Phần 1)". Đừng quên chia sẻ tài liệu này để chúng mình có thêm động lực biên soạn các phần tiếp theo của cuốn sách nhé.

Thân,
Ban biên tập.

Thợ lành nghề #1 - Mở đầu Thảm họa	
13 February 2002, Dear Diary,	Ngày 13 tháng 2 năm 2002. Nhật ký thân mến,
Today was a disaster – I really messed it up. I wanted so much to impress the **journeymen** here, but all I did was **make a mess**.	Ngày hôm nay đúng là một thảm hoạ – Tôi đã làm hỏng mọi chuyện. Tôi rất muốn gây ấn tượng với các ngài **"cựu học việc"** ở đây nhưng rốt cuộc chỉ **làm rối tung** cả lên.
It was my first day on the job as an **apprentice** to Mr. C. I was lucky to get this apprenticeship. Mr. C is a **well-recognized** master of **software development**. The competition for this position was **fierce**.	Đó là ngày đầu tiên tôi được làm một chân **thực tập** của ông C. Tôi quả rất may mắn mới có được vị trí này. Ông C là một người **có tiếng** trong làng **phát triển phần mềm**. Cuộc thi để giành được chân học việc này đúng là **nảy lửa**.
Mr. C's apprentices often become journeymen in **high demand**. It means something to have worked with Mr. C.	Những người theo học ông C thường sẽ trở thành những người thợ lành nghề được **đánh giá cao**. Điều này có nghĩa được làm việc với ông C có giá trị rõ ràng.
I thought I was going to meet him today, but instead a journeyman named Jerry took me aside. He told me that Mr. C always puts his apprentices through an **orientation**	Tôi cứ ngỡ hôm nay sẽ được gặp ông C, nhưng tôi bị gã "cựu học việc" Jerry níu qua một bên. Gã bảo ông C luôn luôn yêu cầu **phần định hướng** cho người học việc trong những ngày đầu.

during their first few days. He said this orientation was to introduce apprentices to the practices that Mr. C insists we use, and to the **level of quality** he expects from our code.	Gã nói ông C nhất quyết cho rằng phần định hướng này rất thiết thực với người học việc, và **chất lượng** mã nguồn mà ông mong đợi ở họ.
This excited me greatly. It was an **opportunity** to show them how good a **programmer** I am. So I told Jerry I couldn't wait to start. He responded by asking me to write a simple program for him. He wanted me to use the Sieve of Eratosthenes to calculate **prime numbers**. He told me to have the program, including all **unit tests**, ready for review just after lunch.	Tôi háo hức kinh khủng. Đây là **cơ hội** để tôi cho họ thấy mình là một tay **lập trình** giỏi cỡ nào. Thế là tôi bảo Jerry tôi không chờ được nữa. Gã đáp lại sự háo hức của tôi bằng cách bảo tôi thử viết một chương trình đơn giản. Gã muốn tôi dùng Sàng Eratosthenes (Sieve) để tính các **số nguyên tố**. Gã còn bảo tôi phải chuẩn bị xong chương trình bao gồm trọn bộ các **kiểm thử đơn vị** để soát sau buổi ăn trưa.
This was great! I had almost four hours to **whip together** a simple program like the Sieve. I am determined to do a really **impressive** job. Listing 1 shows what I wrote. I made sure it was well **commented**, and **neatly** formatted.	Thật tuyệt! Tôi có gần 4 giờ đồng hồ để **"xào nấu"** một chương trình đơn giản như Sieve. Tôi quyết tâm làm một chương trình thật **ấn tượng**. Tôi đã viết mã ở Mã dẫn 1. Tôi nắm chắc là chương trình được **chú thích** cẩn thận và trình bày **gọn gàng**.

Mã dẫn 1

```
/**
 * This class generates prime numbers up to a user-specified maximum.
 * The algorithm used is the Sieve of Eratosthenes. &lt;p&gt; Eratosthenes of Cyrene, b.c.
 * 276 BC, Cyrene, Libya; d.c.194 BC,Alexandria. He was the first man to
 * calculate the circumference of the Earth, and was also known for working on
 * calendars with leap years and running the library at Alexandria.&lt;/p&gt; * The
 * algorithm is quite simple: Given an array of integers starting at 2, cross
 * out all multiples of 2. Find the next uncrossed integer, and cross out all of
 * its multiples. * Repeat until you have passed the square root of the maximum
 * value.
 * @authorAlphonse,
```

```
 * @version 13 Feb 2002 atp
 */

import java.util.*;
public class GeneratePrimes {
    /**
     * @param maxValue is the generation limit.
     */
    public static int[] generatePrimes(int maxValue)
    {
        if (maxValue &gt;= 2) { // the only valid case
            // declarations
            int s = maxValue + 1; // size of array
            boolean[] f = new boolean[s];
            int i;
            // initialize array to true.
            for (i = 0; i &lt; s; i++)
            {
                f[i] = true;
            }
            // get rid of known non-primes.
            f[0] = f[1] = false;
            // sieve
            int j;
            for (i = 2; i &lt; Math.sqrt(s) + 1; i++)
            {
                if (f[i]) { // if i is uncrossed, cross its multiples.
                    for (j = 2 * i; j &lt; s; j += i)
                        f[j] = false; // multiple is not prime
                }
            }
            // how many primes are there?
            int count = 0;
            for (i = 0; i &lt; s; i++) {
                if (f[i]) {
                    count++; // bump count.
                }
            }
            int[] primes = new int[count];
            // move the primes into the result.
            for (i = 0, j = 0; i &lt; s; i++) {
                if (f[i]) // if prime
```

```
            {
                primes[j++] = i;
            }
        }
        return primes; // return the primes.
    } else // maxValue &lt; 2
    {
        return new int[0];  // return null array if bad input.
    }
  }
}
```

Then I wrote a unit test for GeneratePrimes. It is shown in Listing 2. It uses the JUnit framework as Jerry had **instructed**. It takes a **statistical** approach; checking to see if the generator can generate primes up to 0, 2, 3, and 100.	Sau đó tôi viết một kiểm thử đơn vị cho GeneratePrimes. Xem ở mã ở Mã dẫn 2. Đoạn mã này dùng khung làm việc JUnit như Jerry đã **hướng dẫn**. Nó tiếp cận kiểu **thống kê**; kiểm tra xem cái "generator" có thể tạo ra các số nguyên tới 0, 2, 3 và 100.
In the first **case** there should be no **primes**. In the second there should be one prime, and it should be 2. In the third there should be two primes and they should be 2 and 3. In the last case there should be 25 primes, the last of which is 97.	Trong **trường hợp** thứ nhất hẳn không có **số nguyên** nào cả. Trong trường hợp thứ hai phải có một số nguyên và nó phải là số 2. Trường hợp thứ ba có hai số nguyên và chúng là số 2 và 3. Trường hợp cuối phải là 25 số nguyên và số cuối phải là 97.
If all these tests pass, then I **assumed** that the generator was working. I doubt this is foolproof, but I couldn't think of a reasonable **scenario** where these tests would pass and yet the function would fail.	Nếu các kiểm thử đều đạt, thì tôi **giả định** là "generator" chạy. Tôi e rằng khó có thể tin cậy tuyệt đối cách ở trên, nhưng tôi không nghĩ ra được một **kịch bản** có thể xảy ra nào mà một hàm sai nhưng các bước kiểm thử đúng.

Mã dẫn 2

```
import junit.framework.*;
import java.util.*;
public class TestGeneratePrimes extends TestCase {
   public static void main(String args[]) {
                                                junit.swingui.TestRunner.main(new
String[]{&amp;quot;TestGeneratePrimes&amp;quot;});
```

```
    }
    public TestGeneratePrimes(String name) {
        super(name);
    }
    public void testPrimes() {
        int[] nullArray = GeneratePrimes.generatePrimes(0);
        assertEquals(nullArray.length, 0);
        int[] minArray = GeneratePrimes.generatePrimes(2);
        assertEquals(minArray.length, 1);
        assertEquals(minArray[0], 2);
        int[] threeArray = GeneratePrimes.generatePrimes(3);
        assertEquals(threeArray.length, 2);
        assertEquals(threeArray[0], 2);
        assertEquals(threeArray[1], 3);
        int[] centArray = GeneratePrimes.generatePrimes(100);
        assertEquals(centArray.length, 25);
        assertEquals(centArray[24], 97);
    }
}
```

I got all this to work in about an hour. Jerry didn't want to see me until after lunch, so I spent the rest of my time reading the Design Patterns book that Jerry gave me.	Tôi làm việc này mất khoảng một giờ đồng. Jerry không muốn gặp tôi trước bữa ăn trưa, bởi thế, tôi dành trọn bộ thời gian còn lại đọc cuốn Design Patterns mà Jerry đưa cho.
After lunch I stopped by Jerry's office, and told him I was done with the program. He looked up at me with a funny **grin** on his face and said: "Good, let's go **check it out**."	Sau buổi ăn trưa, tôi ghé văn phòng của Jerry và cho gã biết tôi đã thực hiện xong chương trình. Gã nhìn tôi và **cười**, hắn nói: "Được lắm, hãy **xem thử** nó thế nào."
He took me out into the lab and sat me down in front of a workstation. He sat next to me. He asked me to bring up my program on this machine. So I navigated to my laptop on the network and brought up the source files.	Sau bữa trưa, gã dẫn tôi vào **phòng máy** và cho tôi ngồi trước một máy. Gã ngồi bên cạnh tôi và yêu cầu tôi đưa chương trình của tôi vào máy này. Thế là tôi kết nối laptop với mạng và chuyển mã nguồn lên.

Jerry looked at the two source files for about five minutes. Then he **shook his head** and said: “You can’t show something like this to Mr. C! If I let him see this, he’d probably **fire** both of us. He’s not a very **patient** man.”	Jerry xem xét hai mã nguồn chừng năm phút rồi gã **lắc đầu** và bảo: “Mày không thể đưa những cái này cho ông C xem được! Nếu tao để ông ấy xem mấy cái này, thì ông sẽ sa thải cả hai. Ông ấy không phải là người **kiên nhẫn** đâu.”
I was **startled**, but managed to **keep my cool** enough to ask: “What’s wrong with it?” Jerry **sighed**. “Let's **walk through** this together,” he said. “I’ll show you, point by point, how Mr. C wants things done.”	Tôi **đánh thót** một cái nhưng cố **giữ bình tĩnh** và hỏi gã: “Vậy nó sai chỗ nào?” Jerry **thở dài** và nói: “Tụi mình sẽ **lược qua** đống này. Tao sẽ hướng dẫn chi tiết cho mày cách ông C muốn.”
“It seems pretty **clear,**” he continued, “that the main function wants to be three **separate** functions. The first **initializes** all the variables and sets up the sieve. The second actually **executes** the sieve, the third **loads** the sieved results into an **integer array**.”	“Khá là **rõ ràng**”, gã tiếp tục, “rằng nên tách hàm main thành ba hàm **riêng biệt**. Hàm thứ nhất **khởi tạo** tất cả các biến và thiết lập cái “sieve”. Hàm thứ hai thực sự **thi hành** cái “sieve” và hàm thứ ba **tải** kết quả của “sieve” vào một **dãy số nguyên**.”
I could see what he meant. There were three **concepts** which were **buried** in that function. Still, I didn’t see what he wanted me to do about it. He looked at me for a while, clearly expecting me to do something. But finally he **heaved a sigh**, shook his head, and...	Tôi nhận ra được ý gã. Có ba **khái niệm chôn** trong cái hàm đó. Tuy vậy, tôi không biết gã muốn tôi phải làm gì. Gã nhìn tôi một lúc, rõ ràng là đang đợi xem tôi phản ứng sao. Nhưng rốt cuộc gã **thở dài**, **lắc** đầu và….
Thợ lành nghề #2 - Chế độ ăn kiêng tăng cường	
and continued. “To **expose** these three **concepts** more clearly, I want	và tiếp tục. “Để **diễn tả** rõ hơn ba **khái niệm** này, tao muốn mày **tách** chúng

you to **extract** them into three separate methods. Also **get rid** of all the **unnecessary** comments and pick a better name for the **class**. When you are done with that, make sure all the tests still run.”	ra thành ba hàm riêng biệt. Đồng 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.”
You can see what I did in Listing 3. I’ve marked my changes **in bold**, just like Martin Fowler does in his **Refactoring** book. I changed the name of the class to a noun, got rid of all the comments about Eratosthenes, and made three methods out of the three concepts in the generatePrimes function.	Các bạn có thể thấy những gì tôi đã làm trong Mã dẫn 3. Tôi **in đậm** để đánh dấu những thay đổi, tương tự như Martin Fowler trình bày trong cuốn **Tái cấu trúc**. 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.
Extracting the three functions **forced** me to promote some of the **variables** of the function to **static fields** of the class. Jerry said that this made it much **clearer** which variables are **local** and which have wider **influence**.	Việc tách thành ba hàm **bắt** 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

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 told me that this was a little **messy**, so he took the keyboard and showed me how to **clean it up**.	Jerry bảo tôi mã nguồn này hơi **lộn xộn**, nên gã giành lấy bàn phím và chỉ cho tôi cách **dọn dẹp**.
Listing 4 shows what he did. First he **got rid of** the s variable in initializeSieve and replaced it with f.length. Then he changed the names of the three functions to something he said was a bit more **expressive**. Finally he rearranged the **innards** of initializeArrayOfIntegers (née initializeSieve) to be a little nicer to read. The tests all still ran.	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.

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;
      }
   }
```

I had to admit, this was a bit cleaner. I'd always thought it was a waste of time to give functions long **descriptive** names, but his changes really did make the code more **readable**.	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**.
Next Jerry pointed at crossOutMultiples. He said he thought the if(f[i] == true) statements could be made more readable. I thought about it for a minute. The intent of those statements was to check to see if i was uncrossed; so I changed the name of f to unCrossed.	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 said that this was better, but still wasn't pleased with it because it led to **double negatives** like unCrossed[i] = false. So he changed the name of the array to isCrossed and changed the sense of all the booleans. Then he ran all the tests.	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 got rid of the **initialization** that set isCrossed[0] and isCrossed[1] to true. He said it was good enough to just make sure that no part of the function used the isCrossed array for **indexes** less than 2. The tests all still ran.	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 extracted the **inner loop** of the crossOutMultiples function and called it crossOutMultiplesOf. He said that statements like if (isCrossed[i] == false) were **confusing** so he created a function called notCrossed and changed the if statement to if (notCrossed(i)). Then he ran the tests.	Jerry tách **phần lặp bên trong** của hàm crossOutMultiples ra và 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.
Then Jerry asked me what that square root was all about. I spent a bit of time writing a comment that tried to explain why you only have to **iterate up** to the square root of the array size.	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.
I tried to **emulate** Jerry by extracting the calculation into a function where I could put the **explanatory** comment. In writing the comment I realized that the square root is the maximum prime factor of any of the integers in the array. So I chose that name for the variables and functions that **dealt with** it.	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 xử lý vấn đề này
Finally, I made sure that the tests all	Cuối cùng tôi bảo đảm các kiểm thử

still ran. The result of all these changes are shown in Listing 5.	vẫn chạy. Kết quả của các thay đổi trong 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;
}
```

I was starting to **get the hang of** this so I took a look at the putUncrossed-IntegersIntoResult method. I saw that this method had two parts.	Tôi bắt đầu **nắm bắt** được vấn đề nên liền xét hàm putUncrossedIntegersIntoResult. Tôi thấy hàm này có hai phần.
The first counts the number of **uncrossed** integers in the array, and creates the result array of that size. The second moves the uncrossed integers into the result array.	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.
So, as you can see in Listing 6, I extracted the first part into its own function and did some **miscellaneous** cleanup. The tests all still ran. Jerry	Bởi thế, như bạn thấy trong Mã dẫn 6, tôi tách phần thứ nhất ra để hình thành hàm cho chính nó và dọn dẹp

was just **barely** nodding his head. Did he actually like what I did?	lại **một ít**. Các kiểm thử vẫn chạy. Jerry chỉ **thoáng** gật đầu. Gã có thật sự khoái những điều tôi đã thực hiện không?
Thợ lành nghề #3:Tính rõ ràng và cộng tác	
Next Jerry made pass over the whole program, reading it from beginning to end, rather like he was reading a **geometric** proof. He told me that this was a really important step. “So far”, he said, “We’ve been **refactoring fragments**. Now we want to see if the whole program **hangs together** as a readable whole.”	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. “Đến bước này, tụi mình đã thực hiện **tái cấu trúc** 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.”
I asked: “Jerry, do you do this with your own code too?”	Tôi hỏi: “Jerry, anh cũng làm như thế với chính mã nguồn của mình sao?”
Jerry **scowled**: “We work as a team around here, so there is no code I call my own. Do you consider this code yours now?”	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?”
“Not anymore.” I said, **meekly**. “You’ve had a big influence on it.”	Tôi trả lời **nhỏ nhẹ**: “Hết nghĩ như vậy rồi, anh có ảnh hưởng rất lớn đến mã nguồn này.”
“We both have.” he said, “And that’s the way that Mr. C likes it. He doesn’t want any single person owning code.	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.
But to answer your question: Yes. We all practice this kind of refactoring and code clean up around here. It’s Mr. C’s way.”	Câu trả lời cho câu hỏi của mày: “Đúng vậy, ở đây tụi tao thực hành tái cấu trúc và dọn rác. Đó là cách làm của ông C.”

During the **read-through**, Jerry realized that he didn't like the name initializeArrayOfIntegers.	Trong khi **đọc lướt** mã nguồn, Jerry nhận thấy gã không thích cái tên initializeArrayOfIntegers.
"What's being initialized is not, in fact, an array of integers;", he said, "it's an array of booleans. But initializeArrayOfBooleans is not an **improvement**.	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.
What we are really doing in this method is uncrossing all the **relevant** integers so that we can then cross out the **multiples**."	Điều 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)).
"Of course!" I said. So I **grabbed** the keyboard and changed the name to uncross- IntegersUpTo. I also realized that I didn't like the name isCrossed for the array of booleans. So I changed it to crossedOut. The tests all still run. I was starting to enjoy this; but Jerry showed no sign of **approval**.	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.**
Then Jerry turned to me and asked me what I was smoking when I wrote all that maxPrimeFactor stuff. (See Listing 6.) At first I was **taken aback**. But as I looked the code and comments over I realized he had a point. Yikes, I felt stupid! The **square root** of the size of the array is not	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

necessarily prime. That method did not **calculate** the maximum prime factor. The explanatory comment was just wrong. So I **sheepishly rewrote** the comment to better explain the **rationale** behind the square root, and renamed all the variables **appropriately**. The tests all still ran.	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;
    }
}
```

“What the devil is that +1 doing in there?” Jerry **barked** at me.	“dùng +1 ở đây làm quái gì vậy?” Jerry **tru tréo** lên.
I **gulped**, looked at the code, and finally said: “I was afraid that a **fractional** square root would convert to an integer that was one too small to serve as the **iteration limit**.”	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.”
“So you’re going to **litter** the code with extra **increments** just because you are **paranoid**?” he asked. “That’s **silly**, get rid of the increment and run	Gã bèn hỏi: “Cho nên mày **xả rác** trong đoạn mã với **phần “+1”** bởi vì mày bị **hoảng**? Như thế thì ngốc quá,

the tests."	dẹp ngay cái trò gia tăng "+1″ đó và chạy kiểm thử lại đi."
I did, and the tests all ran. I thought about this for a minute, because it made me **nervous**. But I decided that maybe the true iteration limit was the largest prime less than or equal to the square root of the size of the array.	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 **lo lắng**. 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.
"That last change makes me pretty nervous." I said to Jerry. "I understand the rationale behind the square root, but I've got a **nagging** feeling that there may be some corner cases that aren't being **covered**."	"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 chưa **bao quát** hết."
"OK," he **grumbled**. "So write another test that checks that."	Gã **lầm bầm** "OK, vậy thì viết một cái kiểm thử khác để kiểm tra chuyện đó đi."
"I suppose I could check that there are no multiples in any of the prime lists between 2 and 500."	"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, if it'll make you feel better, try that," he said. He was clearly becoming **impatient**.	"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**.
So I wrote the testExhaustive function shown in Listing 8. The new test passed, and my fears were **allayed**.	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.**
Then Jerry **relented** a bit. "It's always good to know why something works," said Jerry. "and it's even better when you show you are right with a test."	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.”
Then Jerry **scrolled** one more time through all the code and tests (shown in listings 7 and 8). He sat back, thinking for a minute, and said: “OK, I think we’re done. The code looks reasonably clean. I’ll show it to Mr. C.”	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.”
Then he looked me **dead in the eye** and said: “Remember this. From now on when you write a **module**, get help with it and keep it clean. If you hand in anything below those standards you won’t **last long** here.” And with that, **he strode off**.	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);
        }
    }
}
```

What a disaster! I thought that my original solution had been **top-notch**.	Quả là tai hoạ! Tôi cứ ngỡ là giải pháp nguyên thủy của tôi là **thượng hạng**.

In some ways I still feel that way. I had tried to show off my **brilliance**, but I guess Mr. C values **collaboration** and **clarity** more than **individual** brilliance.	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**.
I had to admit that the program reads much better than it did at the start. It also works a bit better. I was pretty pleased with the **outcome**. Also, in spite of Jerry's **gruff attitude**, it had been fun working with him. I had learned a lot.	Tôi phải thú nhận rằng chương trình này dễ xem hơn lúc khởi đầu. Nó lại làm việc tốt 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
Still, I was pretty **discouraged** with my performance. I don't think the **folks** here are going to like me very much. I'm not sure they'll ever think I'm good enough for them. This is going to be a lot harder than I thought.	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.
Thợ lành nghề #4 - Bài kiểm tra tính kiên nhẫn	
12 July 2002 Dear Diary, Last night I stared out the window for hours watching the stars **drifting** through the night sky. I felt **conflicted** about the work I did with Jerry yesterday. I learned a lot from working with Jerry on the prime generator, but I don't think I impressed him very much. And, **frankly**, I wasn't all that impressed with him. He spent a lot of time **polishing** a piece of code that worked just fine.	Ngày 12 tháng 7 năm 2002 Nhật ký thân yêu, Tối qua tôi ngồi tựa vào cửa sổ hàng giờ, nhìn các vì sao **trôi dạt** trên bầu trời đêm. Tôi thấy việc làm của tôi và Jerry hôm qua có nhiều **xung đột**. Tôi học hỏi rất nhiều trong khi làm việc với Jerry với vấn đề tạo số nguyên tố, nhưng tôi không tin tôi gây ấn tượng gì với gã. Và, **thật tình mà nói**, tôi cũng không nể gã cho lắm. Thật ra, gã tốn khá nhiều thời gian **mài dũa** các đoạn mã cho dù những đoạn này làm việc ngon lành.

Today Jerry came to me with a new exercise. He asked me to write a program that calculates the **prime factors** of an integer. He said he'd work with me from the start. So the two of us sat down and began to program.	Hôm nay Jerry đến gặp tôi với một bài tập mới. Gã yêu cầu tôi viết một chương trình tính **thừa số nguyên tố** của số nguyên. Gã cho biết gã sẽ làm việc với tôi ngay từ đầu nên hai chúng tôi ngồi xuống và bắt đầu lập trình.
I was pretty sure I knew how to do this. We had written the prime generator yesterday. Finding prime factors is just a matter of walking through a list of primes and seeing if any are factors of the given integer.	Tôi tin chắc tôi biết cách làm. Hôm qua chúng tôi đã viết chương trình tạo số nguyên tố. Dò tìm các thừa số nguyên tố chỉ là vấn đề đi xuyên qua danh sách các số nguyên tố và xét thử có thừa số nào từ các số nguyên đã định.
So I grabbed the keyboard and began to write code. After about half an hour of writing and testing I had produced the following.	Thế nên tôi với lấy bàn phím và bắt đầu viết mã. Khoảng nửa giờ sau khi viết và kiểm tra, tôi làm được như sau:

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;

public class PrimeFactorizer {

   public static void main(String[] args) {
      int[] factors = findFactors(Integer.parseInt(args[0]));
      for (int i = 0; i &amp;amp;lt; factors.length; i++) {
         System.out.println(factors[i]);
      }
   }

   public static int[] findFactors(int multiple) {
      List factors = new LinkedList();
      int[] primes = PrimeGenerator.generatePrimes((int) Math.sqrt(multiple));
      for (int i = 0; i &amp;amp;lt; primes.length; i++) {
         for (; multiple % primes[i] == 0; multiple /= primes[i]) {
```

```
                factors.add(new Integer(primes[i]));
            }
        }
        return createFactorArray(factors);
    }

    private static int[] createFactorArray(List factors) {
        int factorArray[] = new int[factors.size()];
        int j = 0;
        for (Iterator fi = factors.iterator(); fi.hasNext();) {
            Integer factor = (Integer) fi.next();
            factorArray[j++] = factor.intValue();
        }
        return factorArray;
    }
}
```

I tested the program by running the main program with several different **arguments**. They all seemed to work. Running it with 100 gave me 2, 2, 5, and 5. Running it with 32767 gave me 7, 31, and 151. Running it with 32768 gave me fifteen twos.	Tôi kiểm tra chương trình bằng cách chạy nó với nhiều **tham số** khác nhau. Mọi thứ dường như ổn thoả. Chạy chương trình với giá trị tham số là 100 cho tôi kết quả 2, 2, 5 và 5. Chạy nó với 32767 cho tôi 7, 31 và 151. Chạy với 32768 cho tôi 15 số 2.
Jerry just sat there and watched me. He didn't say a word. This made me nervous, but I kept on **massaging** and testing the code until I was happy with it. Then I started writing the unit tests.	Jerry ngồi nhìn tôi. Gã chẳng nói nửa lời. Điều này làm tôi hơi hoảng nhưng tôi tiếp tục **nắn bóp** và thử nghiệm mã nguồn cho đến lúc tôi hài lòng. Sau đó, tôi bắt đầu viết phần kiểm thử đơn vị.
"What are you doing?" asked Jerry. "The program works, so I'm writing the unit tests." I replied. "Why do you need unit tests if the program already works?" he said?	Jerry hỏi: "Mày làm gì vậy?" "Chương trình chạy nên tôi đang viết các kiểm thử đơn vị." Tôi đáp lại. "Nếu chương trình đã chạy thì việc gì mày cần kiểm thử đơn vị?" Gã hỏi

	tiếp.
I hadn't thought of it that way. I just knew that you were supposed to write unit tests. I **ventured** a guess: "So that other programmers can see that it works?"	Tôi không nghĩ đến điểm này. Tôi chỉ biết theo thông lệ cần phải viết kiểm thử đơn vị. Tôi **liều lĩnh** đoán mò: "Để các lập trình viên khác biết được là chương trình đó chạy?"
Jerry looked at me for about thirty seconds. Then he shook his head and said: "What are they teaching you guys in school nowadays?"	Jerry nhìn tôi khoảng 30 giây rồi gã lắc đầu và nói: "Thời buổi này họ dạy dỗ tụi mày cái gì ở trường vậy?"
I started to answer, but he stopped me with a look.	Tôi cứng lưỡi không trả lời được nhưng gã ngăn tôi lại bằng một cái nhìn.
"OK", he said, "delete what you've done. I'll show you how we do things around here."	"OK", gã nói, "xoá hết những thứ mày đã làm đi. Tao chỉ cho mày cách tụi tao làm ở đây."
I wasn't prepared for this. He wanted me to delete what I had just spent thirty minutes creating. I just sat there in **disbelief**.	Tôi quả không chuẩn bị cho tình thế như vậy. Gã muốn tôi xoá những gì tôi đã tạo ra trong ba mươi phút qua. Tôi chỉ ngồi yên, **hoài nghi** những gì mình vừa nghe.
Finally, Jerry said: "Go ahead, delete it."	Cuối cùng Jerry nói: "Xoá đi."
"But it works." I said.	Tôi trả lời: "Nhưng chương trình đó chạy mà."
"So what?" Said Jerry.	"Thì sao?" Jerry đáp lại.
I was starting to get **testy.** "There's nothing wrong with it!" I **asserted**.	Tôi **cứng cổ** cãi. Tôi **khẳng khái**: "Chương trình này chẳng có gì sai hết!"
"Really." he grumbled ; and he grabbed the keyboard and deleted my code.	"Thực vậy hở?" gã lầm bầm và vớ lấy bàn phím, rồi xoá hết mã nguồn của tôi.

I was **dumbfounded**. No, I was **furious.** He had just **reached over** and deleted my work. For a minute I stopped caring about the **prestige** of being an **apprentice** of Mr. C.	Tôi **chết lặng**. Không phải, tôi **điên tiết** lên. Gã mới vừa **chồm qua** và xoá hết đồ của tôi. Trong phút chốc ấy tôi chẳng còn thiết gì đến **ưu thế** được làm một tay **học việc** cho ông C nữa.
What good was that apprenticeship if it meant I had to work with **brutes** like Jerry? These, and other less **complimentary, thoughts raced** behind my eyes as I **glared** at him.	Học việc mà phải đụng đến những kẻ **tàn bạo** như Jerry thì còn hay ho gì nữa? Với ý nghĩ như thế và những **ý nghĩ** còn kém phần **tưởng thưởng** khác **nhảy ra** trong đầu khi tôi **nhìn gã chằm chặp**.
“Ah. I see that upsets you.” Jerry said **calmly**.	“À, tao thấy mày nổi đoá rồi đó.” Jerry nói một cách **điềm tĩnh.**
I **sputtered**, but couldn’t say anything **intelligent**.	Tôi **lắp bắp** nhưng chẳng thốt được gì ra hồn (thông minh).
“Look.” Jerry said, clearly trying to calm me down. “Don’t become **vested** in your code. This was just thirty minutes worth of work. It’s not that big a deal. You need to be ready to **throw away** a lot more code than that if you want to become any kind of a programmer. Often the best thing you can do with a **batch of** code is throw it out.”	“Này.” Jerry nói, rõ ràng đang cố làm dịu tôi xuống. “Đừng có **bám rịt** mã nguồn của mày quá như vậy. Chỉ có ba mươi phút làm việc mà thôi, chẳng phải là cái gì ghê gớm đâu. Mày phải chuẩn bị tinh thần **vứt bỏ** thêm cả đống mã nguồn nữa nếu mày muốn trở thành một thứ lập trình viên gì đó. Vứt bỏ được **hàng đống** mã nguồn thường là điều tốt nhất mà mày nên làm.”
“But that’s such a waste!” I **blurted**.	Tôi **buột miệng**: “Nhưng làm như thế thì rất phí!”
“Do you think the **value** of a program is in the code?” he asked. “It’s not. The value of a program is in your head.”	Gã hỏi lại: “Bộ mày nghĩ **giá trị** của chương trình nằm trong mã nguồn sao? Không phải vậy. Giá trị của một

	chương trình nằm trong cái đầu của mày đó.”
He looked at me for a second, and then went on. “Have you ever **accidentally** deleted something you were working on? Something that took a few days of **effort**?”	Gã nhìn tôi chừng một giây rồi tiếp tục. “Có bao giờ mày **lỡ tay** xoá cái gì đó đang làm chưa? Cái gì đó mà mày phải mất vài ngày **cố gắng** để làm?”
“Once, at school.” I said. “A **disk** crashed and the latest backup was two days old.	“Có một lần, ở trường”. Tôi nói “**Cái đĩa** bị hỏng và bản lưu trữ thì cách đó hai ngày rồi.”
He **winced** and **nodded knowingly**. Then he asked: “How long did it take you to **recreate** what you had lost?”	Gã **cau mày gật đầu** biểu lộ sự **thông cảm** rồi hỏi: “Mày mất bao lâu để **làm lại** những cái đã bị mất?”
“I was pretty familiar with it, so it only took me about half a day to recreate it.”	“Tôi nắm khá rõ những cái bị mất nên chỉ mất có nửa ngày để làm lại.”
“So you didn’t really lose two days worth of work.”	“Ra thế mày thật sự chẳng mất khối lượng công việc của hai ngày.”
I didn’t care for his logic. I couldn’t **refute** it, but I didn’t like it. It **felt** like I had lost two days worth of work!	Tôi chẳng màng gì đến cái lý luận của gã. Tôi không **bác bỏ** được nhưng tôi không khoái cái lý luận đó. Chỉ đơn giản là tôi **cảm thấy** bị mất một khối lượng hai ngày làm việc!
“Did you notice **whether** the new code was better or worse than the code you lost?” he asked.	Gã hỏi tiếp: “Mày có để ý **liệu** phần mã làm lại tốt hay tệ hơn phần mã mày mất không?”
“Oh, it was much better.” I said, **regretting** my words the **instant** I said them. “I was able to use a much better **structure** the second time.”	“Ồ, tốt hơn nhiều.” Tôi nói, **hối tiếc** ngay **giây phút** tôi thốt ra. “Lần thứ hai tôi có thể dùng một **cấu trúc** tốt hơn nhiều.”
He smiled. “So for an extra 25% effort, you **wound up** with a better solution.”	Gã cười. “Thế thì cố thêm 25%, mày **đưa đến** một giải pháp tốt hơn.”

His logic was annoying me. I shook my head and nearly shouted: “Are you suggesting that we always throw away our code when we are done?”	Lý luận của gã làm tôi bực mình. Tôi lắc đầu và gần như thét lên: “Có phải ông giả định là chúng ta luôn luôn vứt bỏ mã nguồn sau khi làm xong?”
To my **astonishment** he nodded his head and said: “Almost. I’m suggesting that throwing away code is a **valid** and **useful** operation. I’m suggesting that you should not **view** it as a loss. I’m **suggesting** that you not get vested in your code.”	Trả lời cho sự **kinh ngạc** của tôi, gã gật đầu và nói: “Gần như là như vậy. Tao giả định chuyện vứt bỏ mã nguồn là một việc **đáng giá** và **hữu dụng**. Tao giả định mày không nên **xem** đó là chuyện hoang phí. Tao **giả định** mày không nên ôm khư khư cái mã nguồn của mày.”
Thợ lành nghề #5: Bước nhỏ	
I didn’t like this; but I didn’t have an argument to use against him. I just sat there in **silent disagreement**.	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**.
“OK”, he said, “Let's start over. The way we work around here is to write our unit tests first. “	“Được rồi”, gã nói, “Mình làm lại từ đầu. Cách tụi tao làm ở đây là viết kiểm thử đơn vị trước”.
This was **patently absurd**. I **reacted** with intelligence: “Huh?”	Điều này **hẳn là vô lý**. Tôi nhanh trí **phản ứng** ngay: “Hở?”
“Let me show you,” he said. “Our task is to create an array of prime factors from a single integer.	“Để 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.
What is the simplest test case you can think of?”	Mày nghĩ được trường hợp kiểm thử nào đơn giản nhất?”
“The first valid case is 2. And it should **return** an array with just a single 2 in it.”	“Trường hợp hợp lý đầu tiên là 2. Và kết quả cần **trả về** một mảng với chỉ một số 2.”
“Right,” he said. And he wrote the	“Đúng rồi.” Gã nói. Và gã viết một

following unit test.	kiểm thử đơn vị như sau:

```
    int factors[] = PrimeFactorizer.factor(2);
        assertEquals(1, factors.length);
        assertEquals(2, factors[0]);
    }
```

Then he wrote the simplest code that would allow the test case to **compile**.	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];
       }
    }
```

He ran the test, and it failed saying: "testTwo(TestPrimeFactors): expected: <1> but was: <0>".	Gã chạy kiểm thử và nó báo lỗi: "testTwo(TestPrimeFactors): expected: <1> but was: <0>".
Now he looked at me and he said: "Do the simplest thing possible to make that test case pass."	Đến đây gã nhìn tôi và nói: "Hãy làm cách gì đơn giản nhất để vượt qua trường hợp kiểm thử đó."
This was **absurdity** upon absurdity. "What do you mean?" I said. "The **simplest** thing would be to return an array with a 2 in it."	Bây giờ thì chồng chất **sự vô lý**. "Ý ông là sao?" Tôi hỏi. "Điều **đơn giản nhất** hẳn trả về một mảng với số 2 trong đó."
With a straight face, he said, "OK, do that."	Gã trả lời với vẻ mặt nghiêm nghị: "Ừ, làm vậy đi."
"But that's silly." I said. "It's the wrong code. The real solution isn't going to just return a 2."	"Nhưng ngớ ngẩn quá." Tôi nói, "Cái mã này sai. Giải pháp thực sự không chỉ trả về có số 2."

“Yes, that’s true.” he said, “But just **humor** me for a bit.”	“Ừa, đúng vậy.” Gã đáp lời. “Nhưng **chiều lòng** tao một chút đi.”
I sighed, rolled my eyes, **huffed and puffed** a bit, and then wrote:	Tôi thở dài bực dọc, đảo mắt nhìn gã, **thỏ dốc** một chút rồi bắt đầu viết:

```
public static int[] factor(int multiple) {
   return new int[]{2};
}
```

I ran the tests, and – of course – they passed.	Tôi chạy cái kiểm thử và tất nhiên nó ổn cả.
“What did that **prove**?” I asked.	Tôi hỏi “Cái này **chứng minh** được điều gì vậy?”
“It proved that you could write a function that finds the prime factors of two.” He said. “It also proves that the test passes when the function **responds** correctly to two.”	“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.”
I rolled my eyes again. This was **beneath** my intelligence. I thought being an apprentice here was supposed to teach me something.	Tôi đảo mắt lần nữa. Mấy thứ này **nằm dưới** “trí tuệ” của tôi. Tôi ngỡ làm một tay học việc ở đây sẽ được dạy một cái gì đó cơ chứ.
“Now, what’s the simplest test case we can add to this?” he asked me.	“Bây giờ, trường hợp kiểm thử nào đơn giản nhất mình có thể đưa thêm vào?” Gã hỏi tôi.
I couldn’t help myself. I **dripped** with **sarcasm** as I said: “Gosh, Jerry, maybe we should try a three.”	Tôi không kìm được, tôi **chì chiết** một cách mỉa mai với câu nói: “Ôi, Jerry hay là mình nên thử với số 3?”
And though I expected it, I was also **incredulous**. He actually wrote the test case for three:	Và mặc dù tôi có hy vọng, tôi **không tin** rằng 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]);
}
```

Running it produced the expected **failure**: “testThree(TestPrimeFactors): expected: <3> but was: <2>”.	Cái kiểm thử này thông báo **lỗi** như đã đoán trước: “testThree(TestPrimeFactors): expected: <3> but was: <2>”
“OK, Alphonse, do the simplest thing that will make this test case pass.”	“Được rồi Alphonse, làm cách gì đơn giản nhất để vượt qua cái kiểm thử này.”
Impatiently, I took the keyboard and typed the following:	**Sốt ruột**, tôi với 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};
   }
}
```

I ran the tests, and they passed.	Tôi chạy mấy cái kiểm thử và chúng đều ổn cả.
Jerry looked at me with an **odd** kind of smile. He said: “OK, that passes the tests. However, it’s not very **bright**, is it?”	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?”
He’s the one who started this **nonsense** and now he’s asking me if this is bright? “I think this whole	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

exercise is pretty **dim**.” I said.	tập này khá **lờ mờ** đó.” Tôi nói.
He **ignored** me and continued. “Every time you add a new test case, you have to make it pass by making the code more **general**. Now go back and make the simplest change that is more general than your first solution.”	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.”
I thought about this for a minute. At last Jerry had asked me something that might require a few **brain cells.** Yes, there was a more general solution. I took the keyboard and typed:	Tôi nghĩ về vấn đề này chừng một phút. Rốt cuộc Jerry đã hỏi tôi vài điều cần đến **tế bào não**. Đúng vậy, có giải pháp tổng quát hơn nữa. Tôi với lấy bàn phím và gõ như sau:

```
public static int[] factor(int multiple) {
    return new int[]{multiple};
}
```

The tests passed, and Jerry smiled. But I still couldn’t see how this was getting us any **closer** to generating prime factors. As far as I could tell, this was a **ridiculous** waste of time.	Các kiểm thử đều ổn cả và Jerry mỉm cười nhưng tôi vẫn không thể hình dung làm sao mấy trò này đưa đến chúng tôi đến **gần hơn** với bài toán tạo ra thừa số nguyên tố. Đến mức này điều duy nhất tôi có thể phát biểu là những cái trò **quái đản** này chỉ phí thời gian.
Still, I wasn’t surprised when Jerry asked me: “Now what’s the simplest test case we can add?”	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?”
“Clearly that would be the case for four.” I said impatiently. And I grabbed	“Rõ ràng là cho trường hợp số 4.” Tôi nói một cách thiếu kiên nhẫn rồi vớ lấy

the keyboard and wrote:	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]);
}
```

“I expect the first assert will fail because an array of size 1 will be returned.” I said.	Tôi nói “Tôi nghĩ cái ‘assert’ thứ nhất sẽ hỏng vì sẽ trả về 1 là kích cỡ của mảng.”
Sure enough, when I ran the test it reported: testFour(TestPrimeFactors):expected <2> but was <1>.	Quả vậy, khi chạy kiểm thử cho báo cáo: “testFour(TestPrimeFactors) :expected <2> but was <1>”.
“I **presume** you’d like me to make the simplest **modification** that will make all these tests pass, and will make the factor method more general?” I asked.	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 just nodded.	Jerry gật đầu.
I made a **concerted** effort to solve only the test case **at hand**, ignoring the test cases I knew would be next. This **galled** me, but it was what Jerry wanted. The result was:	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 **khó chịu** 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;
        }
    }
```

This passed all the tests, but was pretty **messy**. Jerry **scrunched up** his face as though he smelled something **rotten**. He said: “We have to refactor this before we go any further.”	Đoạn mã này vượt qua tất cả các kiểm thử, nhưng nhìn khá **lộn xộn**. Jerry **nhăn mặt** như thể gã đánh hơi được gì đó **bị thối.** Gã nói: “Mình phải tái cấu trúc cái này trước khi đi tiếp.”
“Wait.” I **objected**. “I agree that it’s a bit messy. But shouldn’t we get it all working first and then refactor it if there’s time?”	“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?”
“Egad! No!” said Jerry. “We need to refactor it now so that we can see the true structure as it **evolves**. Otherwise we’ll just keep piling mess upon mess, and we’ll lose the sense of what we’re doing.”	“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 ta chỉ chồng chất cái bừa bộn trên cái bừa bộn và chúng ta sẽ không biết mình đang làm gì nữa.”
“OK.” I sighed. “Let's clean this up.”	“Được thôi.” Tôi thở dài. “Thì dọn dẹp.”

So the two of us did a little refactoring. The result follows:	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;
    }
}
```

"Time for the next test case." Said Jerry; and he passed me the keyboard.	Jerry tuyên bố: "Đến lúc cho cái kiểm thử tiếp theo." và gã chuyển bàn phím cho tôi.
I still couldn't see where this was going, but there was no way out of it. **Obligingly,** I typed in the following test case:	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]);
}
```

"That's interesting.", I said as I **stared** at the green bar, "That one works without change."	"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."
"That is interesting." Said Jerry. "Let's try the next test case."	"Đúng là lý thú". Jerry nối tiếp. "Hãy thử với kiểm thử tiếp theo."
Now I was **intrigued**. I hadn't expected the test case to just work. As I thought about it, it was obvious why it worked, but I still hadn't **anticipated** it. I was pretty sure the next test case would fail, so I typed it in and ran it.	Lúc này tôi rõ ràng đã **bị thu hút**. Tôi không **đoán được** các trường hợp kiểm thử làm việc như vậy. 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 = &amp;quot;assertContains:&amp;quot; + n;
    for (int i = 0; i &amp;lt; factors.length; i++) {
        if (factors[i] == n) {
            return;
        }
    }
    fail(error);
}
```

“Yikes! That one passed too!” I cried.	“Úi! Cái kiểm thử này cũng ổn luôn!” tôi rú lên.
“Interesting.” Nodded Jerry. “Seven is going to work too, isn’t it?”	“Lý thú.” Jerry gật gù. “Vậy 7 sẽ chạy luôn phải không?”
“Yeah, I think it is.”	“Vâng, tôi nghĩ vậy.”
“Then let's skip it and go for eight. That one can’t pass!”	“Vậy thì bỏ nó đi và đi thẳng tới 8, nó sẽ không qua được trường hợp này đâu!”
He was right. Eight had to fail because the factorRegister array was too small.	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 = &amp;quot;assertContains(&amp;quot; + n + &amp;quot;,&amp;quot; + f +
&amp;quot;)&amp;quot;;
        int count = 0;

        for (int i = 0; i &amp;lt; factors.length; i++) {
                if (factors[i] == f) {
                        count++;
        }
        }

        if (count != n) {
                fail(error);
        }
}
```

“What a **relief**! It failed!”	“Đúng là **nhẹ nhõm**! nó hỏng rồi!”
“Yeah,” said Jerry, “for an array out of bounds **exception**. You could get it to pass by **increasing** the size of factorRegister, but that wouldn’t be more general.”	“Ừa.” Jerry đáp “Vì vượt quá **ngoại lệ** 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.
“Let’s try it anyway, and then we’ll solve the general problem of the array size.”	Thì cứ thử xem sao rồi mình giải quyết vấn đề chiều dài của mảng sau.”
So I changed the 2 to a 3 in the initialize function, and got a green bar.	Thế là tôi đổi 2 thành 3 trong hàm initialize và tôi có cái thanh màu xanh.
“OK,” I said, “what is the maximum number of factors that a number can have?”	“Được rồi,” tôi nói. “tối đa các thừa số mà một số có thể có là bao nhiêu?”
“I think it’s something like log2 of the number.” said Jerry.	“Tao nghĩ là logarit cơ số 2 của số đó thì phải.” Jerry nói.
“Wait!” I said, “Maybe we’re **chasing**	“Đợi đã!” Tôi nói, “Có thể mình đang

our tail. What is the largest number we can **handle**? Isn't it 2^64?"	**đi lòng vòng đấy**. Số lớn nhất mình có thể **xử lý** là mấy? không phải là 2 mũ 64 sao?"
"I'm pretty sure it can't be larger than that," said Jerry.	Jerry đáp "Tao chắc là không thể lớn hơn con số đó."
"OK, then let's just make the size of the factorRegister 100. That's big enough to **handle** any number we throw at it.	"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."
"Fine by me." said Jerry. "A hundred integers is nothing to worry about."	"Được thôi." Jerry nói "100 số nguyên thì chẳng có gì phải lo."
We tried it, and the tests still ran.	Chúng tôi thử điều này và các kiểm thử vẫn chạy.
I looked at Jerry and said: "The next test case is nine. That's certainly going to fail."	Tôi nhìn Jerry và nói: "kiểm thử tiếp theo của tôi là 9. Chắc chắn nó sẽ hỏng."
"Let's try it." he said.	Gã đáp "Thì thử đi."
So I typed in the following:	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);
}
```

"Good, that failed." I said. "Making it pass should be simple. I just need to remove 2 as a special number in findPrimeFactors, and use both 2 and 3 with some general **algorithm**." So I **modified** findPrimeFactors as follows:	"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 đã **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;
            }
        }
    }
```

"OK, that passes." Said Jerry. "Now what's the next failing test case?"	"Được rồi, nó đã chạy". Jerry nói. "Bây giờ xem thử cái kiểm thử tiếp theo nào sẽ hỏng?"
"Well, the simple algorithm I used will **divide** by non-primes as well as primes. That won't work right. That's why my first version of the program was divided only by primes.	"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.
The first non-prime the algorithm will divide by is four, so I imagine 4X4 will fail.	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);
}
```

"Ouch! That passes." I said. "How could that pass?"	"Ui! Cái kiểm thử này chạy rồi." Tôi nói. "Làm sao nó qua được nhỉ?"
"It passes, because all the twos have been removed before you try to divide by four, so four is never found as a factor. Remember, it also wasn't found as a factor of 8 or 4!"	"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!"

“Of course!” I said. “All the primes are removed before their **composites**. The fact that the algorithm checks the composites is **irrelevant**. But that means I never needed the array of prime numbers that I had in my first version.”	“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.”
“Right.” said Jerry. “That’s why I deleted it.”	“Đúng thế.” Jerry nói. “Đó là lý do tao xoá nó.”
“Is this it then? Are we done?”	“Vậy thì xong? Mình hoàn thành rồi phải không?”
“Can you think of a failing test case?” asked Jerry?	Jerry hỏi: ” Mày có thể nghĩ ra được cái test case nào bị hỏng không?”
“I don’t know.” I said. “Let's try 1000.”	“Tôi không biết nữa, hãy thử 1000 đi.” Tôi trả lời.
“Ah, the shotgun **approach**. OK, give it a try.”	“À, **tiếp cận** kiểu mạnh bạo. 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);
}
```

“That worked! OK, how about...”	“Nó chạy luôn! Được rồi, hay là…”
We tried several other test cases, but they all passed. This version of the program was much simpler than my first version, and was faster too. No	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

wonder Jerry deleted the first one.	chạy nhanh hơn nữa. Hèn chi Jerry đã xoá đi phiên bản đầu.
What **amazed** me, and still amazes me, is that we **snuck up** on the better solution one test case at a time. I don't think I would ever have **stumbled upon** this simple approach had we not been **inching** forward one simple test case at a time.	Điều làm tôi **kinh ngạc** và vẫn còn làm tôi kinh ngạc là sau mỗi kiểm thử chúng tôi lại **tiến gần hơn** với giải pháp. Nếu không **tiến lên** với mỗi kiểm thử thì tôi không nghĩ sẽ có thể **đến** theo cách đơn giản này.
I wonder if that happens in bigger projects? I learned something today.	Tôi không biết chuyện gì sẽ xảy ra với những dự án lớn hơn nữa? Hôm nay tôi đã học được đôi điều.

Từ vựng	Phiên âm	Nghĩa
journeymen	ˈdʒɜː.ni.mən	cựu học việc
make a mess	meɪk ə mes	tạo một mớ hỗn độn
apprentice	əˈpren.tɪs	học nghề
well-recognized	wel ˈrek.əg.naɪzd	được công nhận
software development	ˈsɒft.weər dɪˈvel.əp.mənt	phát triển phần mềm
fierce	fɪəs	căng thẳng
high demand	haɪ dɪˈmɑːnd	Nhu cầu cao
orientation	ˌɔː.ri.enˈteɪ.ʃən	sự định hướng
level of quality	ˈlev.əl əv ˈkwɒl.ə.ti	mức độ chất lượng
opportunity	ˌɒp.əˈtʃuː.nə.ti	cơ hội
programmer	ˈprəʊ.græm.ər	lập trình viên
prime number	praɪm ˈnʌm.bər	số nguyên tố
unit test	ˈjuː.nɪt test	kiểm tra đơn vị
whip together	wɪp təˈgeð.ər	
impressive	ɪmˈpres.ɪv	ấn tượng
commented	ˈkɒm.ent	đã nhận xét
neatly	ˈniːt.li	gọn gàng
instructed	ɪnˈstrʌkt	hướng dẫn
statistical	stəˈtɪs.tɪ.kəl	thống kê
case	keɪs	trường hợp
assumed	əˈsjuːm	giả định
scenario	sɪˈnɑː.ri.əʊ	kịch bản
grin	grɪn	cười toe toét
shook sbd's head	ʃʊk	lắc đầu
fire	faɪər	sa thải
patient	ˈpeɪ.ʃənt	kiên nhẫn
startled	ˈstɑː.təld	giật mình
keep cool	kiːp kuːl	giữ bình tĩnh
sighed	saɪ	thở dài
walk through	wɔːk θruː	đi qua
clear	klɪər	thông thoáng

separate	ˈsep.ər.ət	riêng rẽ
initialize	ɪˈnɪʃ.əl.aɪz	khởi tạo
execute	ˈek.sɪ.kjuːt	thi hành
load	ləʊd	tải
integer array	ˈɪn.tɪ.dʒər əˈreɪ	mảng số nguyên
concept	ˈkɒn.sept	Ý tưởng
buried	ˈber.i	chôn
heaved	hiːv	phập phồng
expose	ɪkˈspəʊz	lộ ra
extract	ɪkˈstrækt	trích xuất
unnecessary	ʌnˈnes.ə.ser.i	không cần thiết
get rid of	get rɪd əv	thoát khỏi
in bold	ɪn bəʊld	in đậm
refactor	riˈfæk.tər	cấu trúc lại
forced	fɔːst	bị ép
variable	ˈveə.ri.ə.bəl	Biến
static field	ˈstæt.ɪk fiːld	trường tĩnh
clearer	ˈklɪərər	rõ ràng hơn
local	ˈləʊ.kəl	địa phương
influence	ˈɪn.flu.əns	ảnh hưởng
messy	ˈmes.i	lộn xộn
clean sth up	kliːn ʌp	dọn dẹp
expressive	ɪkˈspres.ɪv	biểu cảm
innards	ˈɪn.ədz	bộ phận
descriptive	dɪˈskrɪp.tɪv	mô tả
readable	ˈriː.də.bəl	có thể đọc được
double negatives	ˈdʌb.əl ˈneg.ə.tɪv	phủ định kép
initialization	iˌniSHələˈzāSH(ə)n	sự khởi tạo
index	ˈɪn.deks	mục lục
inner loop	ɪn.ər luːp	vòng trong
confusing	kənˈfjuː.zɪŋ	gây nhầm lẫn

iterate up	ˈɪt.ər.eɪt ʌp	lặp lại
emulate	ˈem.jə.leɪt	thi đua
explanatory	ɪkˈsplæn.ə.tər.i	giải thích
dealt with	di:l wɪð	xử lý
get the hang of	get ði: hæŋ əv	hiểu việc
miscellaneous	ˌmɪs.əlˈeɪ.ni.əs	bao hàm
barely	ˈbeə.li	vừa đủ
geometric	ˌdʒi:.əˈmet.rɪk	hình học
fragment	ˈfræg.mənt	miếng
hangs together	hæŋ təˈgeð.ər	gắn bó với nhau
scowl	skaʊl	cau có
meekly	ˈmi:.kli	hiền lành
read-through	ˈri:dˌθru:	Đọc qua
improvement	ɪmˈpru:v.mənt	sự cải tiến
relevant	ˈrel.ə.vənt	liên quan, thích hợp
multiple	ˈmʌl.tɪ.pəl	nhiều
grab	græb	vồ lấy
approval	əˈpru:.vəl	phê duyệt
taken aback	teɪk əˈbæk	sửng sốt
square root	ˌskweə ˈru:t	căn bậc hai
calculate	ˈkæl.kjə.leɪt	tính toán
sheepishly	ˈʃi:.pɪʃ.li	ngượng ngùng
rewrite	ˌri:ˈraɪt	viết lại
rationale	ˌræʃ.əˈnɑ:l	cơ sở lý luận
appropriately	əˈprəʊ.pri.ət.li	thích hợp
bark	bɑ:k	tru tréo
gulp	gʌlp	nuốt nước bọt
fractional	ˈfræk.ʃən.əl	phân số
iteration limit	ˌɪt.ərˈeɪ.ʃən ˈlɪm.ɪt	giới hạn lặp lại
litter	ˈlɪt.ər	xả rác
increment	ˈɪŋ.krə.mənt	tăng

paranoid	ˈpær.ən.ɔɪd	hoang tưởng
silly	ˈsɪl.i	ngốc
nervous	ˈnɜː.vəs	lo lắng
nagging	ˈnæg.ɪŋ	cằn nhằn
covered	ˈkʌv.ər	đề cập
grumbled	ˈgrʌm.bəl	càu nhàu
impatient	ɪmˈpeɪ.ʃənt	nóng nảy
allay	əˈleɪ	xoa dịu
relented	rɪˈlent	mủi lòng
scroll	skrəʊl	cuộn xuống
dead in the eye	ded ɪn ðiːaɪ	lạnh lùng
module	ˈmɒdʒ.uːl	mô-đun
last long	lɑːst lɒŋ	tồn tại lâu
stride off	straɪd ɒf	sải bước
top-notch	ˌtɒpˈnɒtʃ	đỉnh cao
brilliance	ˈbrɪl.jəns	sáng chói
collaboration	kəˌlæb.əˈreɪ.ʃən	sự hợp tác
clarity	ˈklær.ə.ti	trong trẻo
individual	ˌɪn.dɪˈvɪdʒ.u.əl	cá nhân
outcome	ˈaʊt.kʌm	kết cục
gruff	grʌf	cộc cằn
attitude	ˈæt.ɪ.tʃuːd	Thái độ
discouraged	dɪˈskʌr.ɪdʒd	nản lòng
folks	foʊks	hội
drifting	drɪft	trôi dạt
conflicted	kənˈflɪk.tɪd	mâu thuẫn
frankly	ˈfræŋ.kli	thẳng thắn
polish	ˈpɒl.ɪʃ	đánh bóng
argument	ˈɑːg.jə.mənt	tham số
ventured	ˈven.tʃər	mạo hiểm
disbelief	ˌdɪs.bɪˈliːf	sự hoài nghi

testy	ˈtes.ti	cứng rắn
assert	əˈsɜːt	khẳng định
dumbfounded	ˌdʌmˈfaʊn.dɪd	chết lặng
furious	ˈfjʊə.ri.əs	giận dữ
reach over	riːtʃ ˈəʊ.vər	vươn tới
prestige	presˈtiːʒ	uy tín
brute	bruːt	vũ phu
apprenticeship	əˈpren.tɪs.ʃɪp	sự học việc
complimentary	ˌkɒm.plɪˈmen.tər.i	ca ngợi
thought	θɔːt	ý nghĩ
race	reɪs	cuộc đua
glare	gleər	chói mắt
calmly	ˈkɑːm.li	điềm tĩnh
sputtered	ˈspʌt.ər	lắp bắp
intelligent	ɪnˈtel.ɪ.dʒənt	thông minh
vested	ˈvestɪd	bám rịt
throw away	θrəʊ əˈweɪ	vứt đi
batch of	bætʃ əv	lô
blurted	blɜrt	buột miệng
value	ˈvæl.juː	giá trị
accidentally	ˌæk.sɪˈden.təl.i	tình cờ
effort	ˈef.ət	cố gắng
disk	dɪsk	đĩa
winced	wɪns	nhăn mặt
nod	nɒd	gật đầu
knowingly	ˈnəʊ.ɪŋ.li	cố ý
recreate	ˌriː.kriˈeɪt	tái tạo
refute	rɪˈfjuːt	bác bỏ
felt	felt	cảm thấy
whether	ˈweð.ər	liệu
regret	rɪˈgret	hối tiếc

instant	ˈɪn.stənt	lập tức
structure	ˈstrʌk.tʃər	kết cấu
wound up	ˌwaʊnd ˈʌp	quấn lại
astonishment	əˈstɒn.ɪʃ.mənt	sự kinh ngạc
valid	ˈvæl.ɪd	có giá trị
useful	ˈjuːs.fəl	có ích
view	vjuː	Quang cảnh
suggest	səˈdʒest	gợi ý
against	əˈgenst	chống lại
silent	ˈsaɪ.lənt	im lặng
disagreement	ˌdɪs.əˈgriː.mənt	bất đồng ý kiến
patently	ˈpeɪ.tənt.li	nhẹ nhàng
absurd	əbˈsɜːd	ngớ ngẩn
reacted	riˈækt	phản ứng
return	rɪˈtɜːn	trả về
compile	kəmˈpaɪl	biên dịch
absurdity	əbˈzɜː.dɪ.ti	sự vô lý
simple	ˈsɪm.pəl	giản dị
humor	ˈhjuː.mər	chiều lòng
huff and puff	hʌf ænd pʌf	thở hổn hển
prove	pruːv	chứng tỏ
respond	rɪˈspɒnd	trả lời
beneath	bɪˈniːθ	ở trên
drip	drɪp	chỉ chiết
sarcasm	ˈsɑː.kæz.əm	mỉa mai
incredulous	ɪnˈkredʒ.ə.ləs	không tin
failure	ˈfeɪ.ljər	thất bại
Impatiently	ɪmˈpeɪ.ʃənt.li	Sốt ruột
odd	ɒd	bất thường
bright	braɪt	sáng chói
nonsense	ˈnɒn.səns	vô lý

dim	dɪm	lờ mờ
ignore	ɪgˈnɔːr	Làm lơ
general	ˈdʒen.ər.əl	chung
brain cells	breɪn sel	tế bào não
closer	kləʊzer	gần hơn
ridiculous	rɪˈdɪk.jə.ləs	lố bịch
presume	prɪˈzjuːm	phỏng đoán
modification	ˌmɒd.ɪ.fɪˈkeɪ.ʃən	sự sửa đổi
concerted	kənˈsɜː.tɪd	phối hợp
at hand	æt hænd	trong tầm tay
galled	gɔːl	khó chịu
messy	ˈmes.i	lộn xộn
scrunch up	skrʌntʃ ʌp	thu dọn
rotten	ˈrɒt.ən	thối rữa
objected	ˈɒb.dʒɪkt	phản đối
evolve	ɪˈvɒlv	tiến hóa
obligingly	əˈblaɪ.dʒɪŋ.li	mang ơn
stared	steər	nhìn chằm chằm
intrigued	ɪnˈtriːg	bị thu hút
anticipated	ænˈtɪs.ɪ.peɪt	dự đoán trước
relief	rɪˈliːf	sự cứu tế
exception	ɪkˈsep.ʃən	ngoại lệ
increase	ɪnˈkriːs	tăng
chasing tail	tʃeɪs teɪl	đi lòng vòng
handle	ˈhæn.dəl	xử lý
algorithm	ˈæl.gə.rɪ.ðəm	thuật toán
modified	ˈmɒdɪfaɪd	sửa đổi
divide	dɪˈvaɪd	chia
composite	ˈkɒm.pə.zɪt	đa hợp
irrelevant	ɪˈrel.ə.vənt	không liên quan
approach	əˈprəʊtʃ	cách tiếp cận

amazed	əˈmeɪzd	kinh ngạc
sneak up	snik ʌp	lẻn lên
stumble upon	ˈstʌm.bəl əˈpɒn	vấp ngã
inching	ɪntʃ	nhích từng chút

PHỤ LỤC: TÀI NGUYÊN LẬP TRÌNH

Bên cạnh việc học tiếng Anh, bạn có thể tự học lập trình thông qua các tài liệu, khoá học miễn phí do CodeGym cung cấp:

1. Kho tài liệu lập trình miễn phí: https://codegym.vn/tai-nguyen-hoc-lap-trinh/
Tải nguyên học lập trình là trang tổng hợp +200 tài liệu, sách, khoá học, bài thực hành, video hướng dẫn lập trình… từ cơ bản đến nâng cao, đa dạng chủ đề, phù hợp với mọi đối tượng từ các bạn bắt đầu học từ con số 0, cho tới những người đang học/làm lập trình mong muốn nâng cao trình độ, kỹ thuật code…

2. Blog của CodeGym: https://codegym.vn/blog
Đây là nơi tập hợp nhiều bài viết liên quan đến việc học lập trình, bao gồm cả các bài viết kỹ thuật, các bài viết về công nghệ và cả các bài viết định hướng về nghề nghiệp.

3. GitHub của CodeGym https://github.com/codegym-vn
Đây là nơi tập trung các mã nguồn mà CodeGym sử dụng trong quá trình dạy học, các bạn có thể tìm thấy ở đây hàng trăm mã nguồn để tham khảo thuộc các công nghệ khác nhau như Java, PHP, .NET, Javascript, Android, React...

4. Ứng dụng luyện tập CodeGym Bob: https://bob.codegym.vn/home
Đây là ứng dụng rất phù hợp cho những bạn mới bắt đầu đến để luyện tập và khẳng định các năng lực của mình. Ứng dụng này hoàn toàn miễn phí và sẽ tự động giúp bạn đánh giá mức độ thuần thục của mình trong việc áp dụng các kiến thức đã học được.

5. Nhóm Học lập trình: https:// facebook.com/groups/hoclaptrinh.cg
Đây là nơi mà những người mới bắt đầu học lập trình có thể tham gia và thảo luận, nhận được các tư vấn và lời khuyên từ những người đi trước.

6. Các trang web chia sẻ kiến thức hữu ích về các công nghệ
- Học Java: https://hocjava.com
- Học PHP: https://hocphp.net
- Học Laravel: https://hoclaravel.net
- Học Spring MVC: https://hocspringmvc.net
- Học Spring Boot: https://hocspringboot.net
- Học Javascript: https://hocjavascript.net