The Crafsman 1. Opening Diaster.

Robert C. Martin

13 Tháng 2, 2002

Bài viết này lược trích từ chương Principles, Patterns and Practices trong cuốn Agile Software Development của Robert C. Martin, nhà xuất bản Prentice Hall, 2002.

Nhật ký thân mến,

13 tháng 2, 2002.

Hôm nay đúng là một ngày xui xẻo - Tôi làm hỏng cả chuyện. Tôi rất muốn gây ấn tượng với các ngài "cựu học việc" ở đây nhưng rút cuộc chỉ làm rối tung cả lên.

Đó là ngày đầu tiên tôi được một chân học việc với ông C. Tôi quả là may mắn có được chân học việc này. Ông C là một tay trùm lớp lang trong vấn đề phát triển phần mềm. Đấu để giành được chân việc này đúng là nẩy lửa. Các tay học việc của ông C thường trở nên các tay "cựu học việc" sáng giá. Điều này có nghĩa được làm việc với ông C có giá trị rõ ràng.

Tôi cứ ngỡ là hôm nay tôi sẽ được gặp ông ta nhưng thay vì đó tôi bị một gã "cựu học việc" níu tôi qua một bên. Gã bảo ông C luôn luôn dẫn các tay học việc đi xuyên qua phần định hướng trong những ngày đầu. Gã nói ông C nhất quyết cho rằng phần thực tập định hướng là thiết thực với các tay học việc và nó dẫn đến mức chất lượng mã nguồn mà ông ta ta dự tưởng.

Tôi náo nức kinh khủng. Đây là một cơ hội cho họ thấy tôi là một tay lập trình "ngon" cỡ nào. Thế là tôi bảo Jerry tôi không chờ được nữa. Gã đáp lại sự náo nức của tôi bằng cách bảo tôi thử viết một chương trình đơn giản cho gã. Gã muốn tôi dùng "Sieve of Eratosthenes" để tính các số nguyên. Gã còn bảo tôi phải chuẩn bị xong chương trình bao gồm trọn bộ các "unit tests" sẵn sàng để "chấm" sau buổi ăn trưa.

Thật là khoái! Tôi có gần 4 tiếng đồng hồ để "xào nấu" một chương trình giống như Sieve. Tôi quyết tâm thực hiện công tác này một cách hết sức có ấn tượng. Mã dẫn 1 đưa ra những gì tôi đã viết. Tôi nắm chắc là chương trình của tôi 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.
 * <p>
 * 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.</p>
 *
 * 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 >= 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 < s; i++)
      f[i] = true;

    // get rid of known non-primes.
    f[0] = f[1] = false;

    // sieve
    int j;
    for (i = 2; i < Math.sqrt(s) + 1; i++) {
      if (f[i]) { // if i is uncrossed, cross its multiples.
        for (j = 2 * i; j < s; j += i)
          f[j] = false; // multiple is not prime
    }
  }

  // how many primes are there?
  int count = 0;
```

```
    for (i = 0; i < 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 < s; i++) {
      if (f[i]) // if prime
        primes[j++] = i;
    }
    return primes; // return the primes.
  } else // maxValue < 2
    return new int[0];  // return null array if bad input.
  }
}
```

Sau đó tôi viết một cái "unit test" cho GeneratePrimes. Xem ở mã dẫn 2. Đoạn mã này dùng JUnit framework như Jerry đã chỉ dẫn. Nó dùng tính chất hướng thống kê; kiểm tra xem cái "generator" có thể tạo ra các số nguyên tới 0, 2, 3 và 100. Trong trường hợp thứ nhất hẳn không có số nguyên nào cả. Trong trường hợp thứ nhì hẳn phải có một số nguyên và nó phải là số 2. Trường hợp thứ ba phải có hai số nguyên và chúng phải là số 2 và 3. Trường hợp cuối phải là 25 số nguyên và số cuối phải là 97. Nếu các bước kiểm tra đều đúng, tôi giả định là cái "generator" làm việc đúng. Tôi e rằng khó có thể tin cậy tuyệt đối cách ở trên, nhưng tôi không nghĩ ra được một trường hợp nào một "function" có thể bị hỏng mà các bước kiểm tra đều 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[] {"TestGeneratePrimes"});
  }
  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);
  }
}
```

Tôi mất khoảng một giờ đồng hồ để làm những bước trên chạy được. Jerry không muốn gặp tôi cho đến sau buổi ăn trưa, bởi thế, tôi dành trọn bộ thời gian còn lại đọc cuốn Design Patterns mà Jerry đưa cho tôi.

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 với một nụ cười khó tả, hắn nói: "Được lắm, hãy xem thử nó thế nào."

Gã dẫn tôi và phòng thí nghiệm 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 chuyển mã nguồn từ máy laptop của tôi lên.

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 ổng xem mấy cái này, ổng sẽ đuổi cổ cả tao lẫn mày. Ông ấy không phải là người kiên nhẫn đâu."

Tôi đánh thót một phát nhưng cố giữ bình tĩnh và hỏi gã: "Chớ nó sai chỗ nào?"

Jerry thở dài và nói: "Tụi mình nên đi xuyên qua mã nguồn này với nhau. Tao sẽ chỉ cho mày từng điểm một cách ông C muốn thực hiện nó như thế nào."

"Quá rõ ràng", gã tiếp tục, "cái main function muốn làm ra ba cái functions riêng biệt. Cái thứ nhất khởi tạo tất cả các biến hàm và thiết lập cái "sieve". Cái thứ nhì thực sự thi hành cái "sieve" và cái thứ ba tải kết quả của "sieve" vào một dãy số nguyên."

Tôi nhận ra được ý gã muốn nói gì. Có ba khái niệm chôn trong cái function đó. Tuy vậy, tôi không biết gã muốn tôi phải làm gì với nó.

Gã nhìn tôi một lúc, rõ ràng đang đợi tôi phản ứng sao đó. Nhưng rốt cuộc gã thở dài, lắc đầu và....

<đón đọc bài kế tiếp>

The Crafsman 2. Crash Diet.

Robert C. Martin

*Trong phần trước * Jerry, một tay cựu học việc yêu cầu Alphonse, một tay học việc, viết một chương trình tạo các số nguyên dùng "sieve of Etastosthenes". Jerry, nhận thấy Alphonse ứng dụng trọn bộ thuật toán vào một function "khổng tượng" nên đã yêu cầu Alphonse tách nó ra theo ba khái niệm: khởi động, ứng tạo và chuẩn xuất;... 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. "Để mở rộng ba khái niệm rõ ràng hơn, tao muốn mày tách chúng ra thành ba methods riêng biệt. Đồng thời vứt hết những cái phụ chú không cần thiết và đặt một cái tên khá hơn cho cái class. Mày làm xong những thứ đó rồi phải bảo đảm là mấy cái test vẫn còn chạy được."

Các bạn có thể thấy những điểm tôi đã làm trong Mã dẫn 3. Tôi đã đánh dấu những thay đổi bằng chữ đậm, y hệt như Martin Fowler trình bày trong cuốn Refactoring của ông ta. Tôi đổi tên của cái class thành dạng danh từ, vứt hết những phụ chú về chuyện Eratosthenes và tạo ra ba methods từ ba khái niệm trong **generatePrimes** function.

Tách ra ba functions buộc tôi phải đưa ra một số biến hàm của function thành **static fields** của cái class. Jerry nói cách này làm rõ những biến hàm nào là local và những biến hàm nào có ảnh hưởng rộng lớn hơn.

Mã dẫn 3

PrimeGenerator.java, version 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,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,j;
  for (i = 2; i < Math.sqrt(s) + 1; i++) {
    // if i is uncrossed, cross out its multiples.
    if (f[i]) {
      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];

  // initialize array to true.
  for (int i = 0; i < s; i++)
   f[i] = true;

  // get rid of known non-primes
  f[0] = f[1] = false;
  }
 }
```

Jerry bảo tôi mã này hơi lộn xộn, nên gã giành lấy bàn đánh và chỉ tôi cách dọn dẹp. **Mã dẫn 4** minh hoạ những gì gã đã làm. Thoạt tiên gã vứt đi cái biến hàm s trong

initializeSieve và thay thế nó bằng **f.length**. Sau đó gã đổi tên của ba functions (theo kiểu) gã cho là có ấn tượng hơn. Cuối cùng gã sắp xếp lại cái "bộ lòng" **initializeArrayOfIntegers** (từ **initializeSieve**) để cho dễ đọc hơn một chút. Các cái test vẫn chạy nhưng thường.

Mã dẫn 4

PrimeGenerator.java, version 3 (partial)

```
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 mã này rõ hơn một chút. Trước giờ tôi nghĩ tạo functions có tên sinh động là phí thời giờ , nhưng những chỉnh đổi của gã quả thật làm cho mã nguồn dễ đọc hơn.

Tiếp theo Jerry trỏ vào **crossOutMultiples**, nói là gã nghĩ cụm **if(f[i] == true)** có thể làm cho dễ đọc hơn nữa. Tôi nghĩ đến điểm này chừng một phút. Ý định của các cụm này dùng để kiểm tra xem i không bị loại trừ; thế là tôi đổi tên của **f** thành **unCrossed**.

Jerry nói mã này được hơn nhưng tôi vẫn chưa hài lòng với nó vì nó dẫn đến khả năng phủ định đôi (double negative) như **unCrossed[i] = false**. Bởi thế gã đổi tên của dãy số thành dãy **isCrossed** với chỉ số nhỏ hơn 2. Các cái test vẫn chạy được.

Jerry tách phần lặp bên trong (inner loop) của **crossOutMultiples** function và gọi nó là **crossOutMultipleOf**. Gã bảo rằng các cụm tương tự như **if (isCrossed[i] == false)** dễ nhầm lẫn nên gã tạo ra function có tên **notCrossed** và thay cụm **if** thành **if (notCrossed(i))**. Kết tiếp gã chạy thử mấy cái test lại.

Sau đó Jerry hỏi tôi ý nghĩa của phần số căn đó là gì. Tôi tốn ít thời giờ viết phụ chú giải thích tại sao cần phải lặp lại cho đến phần số căn của chiều dài dãy số. Tôi cố tranh đua với Jerry bằng cách tách phần tính toán thành một function, nơi tôi có thể đưa vào phần phụ giải. Trong khi viết phụ chú tôi nhận ra rằng căn số là phân tố cực đại của số nguyên trong một dãy số. Bởi thế để ứng phó, tôi chọn cách gọi đó (maxValue) cho các biến hàm. Cuối cùng tôi bảo đảm các tests vẫn chạy được. Kết quả của các thay đổi trong **Mã dẫn 5**.

Mã dẫn 5

PrimeGenerator.java version 4 (partial)

```
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 primes. 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 lại method **putUncrossedIntegersIntoResult**. Tôi thấy rằng method này có hai phần. Phần thứ nhất đếm các số nguyên không bị loại trong dãy số, và tạo nên dãy kết quả (bằng chiều dài của dãy số). Phần thứ nhì dời các số nguyên không bị loại vào dãy kết quả này. Bởi thế, như bạn thấy trong **Mã dẫn 6**, tôi tách phần thứ nhất ra để hình thành function cho chính nó và dọp dẹp lặt vặt đôi chút. Các tests vẫn chạy được. Jerry chỉ thoáng gật đầu. Gã có thật sự khoái những điều tôi đã thực hiện không?

Mã dẫn 6

PrimeGenerator.java, version 5 (partial).

```
 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;
  }
```

<đón xem phần kế tiếp>

* Trong nguyên bản là "In the last month's column..." nhưng ở đây tạm dịch thoáng ra là "trong phần trước" cho phù hợp với tinh thần các bài craftsman được post lên diễn đàn (không theo tháng mà theo... tùy h ứng của người dịch ;))

The Crafsman 3. Clarity and Collaboration

Rober C. Martin

Lần trước, Jerry, một cựu học việc yêu cầu tay học việc Alphonse viết một chương trình tạo số nguyên tố dùng phương pháp lượt Eratosthenes (sieve of Eratosthenes). Jerry duyệt và giúp Alphonse tách lược (refactor) mã nguồn đó. Anh ta không được hài lòng với kết quả của Alphonse. Lần trước Alphonse thực hiện xong phần refactoring và nghĩ chắc Jerry sẽ chấp thuận...

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

Sau đó Jerry đi xuyên qua trọn bộ chương trình, đọc lại từ đầu đến cuối như thể gã 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 refactoring các mảnh mã. Bây giờ tụi mình xem thử trọn bộ chương trình có thể nối liền nhau như một dạng tổng thể".

Tôi hỏi: "Jerry, bộ ông cũng làm như thế với chính mã nguồn của ông 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 là cái mã này của riêng mày hở?"

Tôi trả lời hết sức nhỏ nhẻ: "hết nghĩ như vậy rồi, ông 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 ưa chuộng. Ông ấy không khoái bất cứ một ai làm chủ mã nguồn hết đâu. Trả lời riêng cho câu hỏi của mày: Đúng vậy, ở đây tụi tao thực nghiệm cái "rơ" refactoring và dọn rác và đây là phương pháp của ông C."

Trong khi đọc qua mã nguồn, Jerry thấy gã không khoái cái tên **initializeArrayOfIntegers**.

Gã nói: "Cái được khởi tạo ở đây thực ra không phải là một dãy số nguyên, mà là một dãy booleans. Nhưng **initializeArrayOfBooleans** không hẳn là cách cải tiến. Điều chúng ta thực sự muốn làm ở method 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 (ie 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!" Thế là tôi vớ lấy bàn đánh và sửa tên của method đó thành **uncrossIntegersUpTo**. Tôi cũng thấy không khoái cái tên **isCrossed** lại dùng cho một dãy booleans, nên tôi đổi nó thành **crossedOut**. Các cái test vẫn chạy. Tôi bắt đầu thấy thích mấy cái trò này nhưng Jerry vẫn không hề tỏ vẻ đồng tình.

Sau đó Jerry quay lại, 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 phụ chú tôi nhận thấy gã có lý. Eo ôi, tôi thấy mình thật là ngu! Căn bậc 2 (Square root)* của chiều dài một dãy số không hẳn là nguyên số. Method đó không tính thừa số nguyên tố cực đại (max prime factor) **. Phần chú giải sai bét nên, hết sức ngượng ngùng tôi viết lại phần phụ chú để giải thích rõ hơn cái căn bậc 2 này dùng để làm gì và đổi tên những biến , hàm cho thích hợp. Các test vẫn chạy.

Mã dẫn 6

TestGeneratePrimes.java (Partial)

```
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;

}
```

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

Tôi nuốt cái ực, 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 gia tăng "+1" bởi vì mày bị hoảng? Như thế thì ngốc quá, dẹp ngay cái trò gia tăng "+1" đó đi và thử test lại."

Tôi làm như thế và trọn bộ các test đều chạy. Tôi suy 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 cực đại" và "thừa số nguyên tố" đó <= căn bậc 2 chiều dài của dãy số.

"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 test 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 cái **testExhaustive** function như trong **Mã dẫn 8**. Phần test 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 cái test."

Sau đó Jerry dò qua trọn bộ mã nguồn và các cái tests một lần nữa (xem **Mã dẫn 7 và 8**). Gã ngã người ra và suy nghĩ chừng một phút rồi nói: "OK, tao nghĩ là tụi mình 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 (clean). Nếu mày nhúng tay vào những thứ dưới tiêu chuẩn này, mày không "thọ" ở đây đâu."

Gã rảo bước.

Mã dẫn 7

PrimeGenerator.java (final)

```
/**
 * 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 < 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 < crossedOut.length; i++)
    crossedOut[i] = false;
}
private static void crossOutMultiples() {
  int limit = determineIterationLimit();
  for (int i = 2; i <= 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 < 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 < crossedOut.length; i++)
      if (notCrossed(i))
        result[j++] = i;
  }
  private static int numberOfUncrossedIntegers() {
    int count = 0;
    for (int i = 2; i < crossedOut.length; i++)
      if (notCrossed(i))
        count++;

    return count;
  }
}
```

Mã dẫn 8

TestGeneratePrimes.java (final)

```
import junit.framework.*;


public class TestGeneratePrimes extends TestCase {
 public static void main(String args[]) {
   junit.swingui.TestRunner.main( new String[] {"TestGeneratePrimes"});
 }
 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<500; i++)
      verifyPrimeList(PrimeGenerator.generatePrimes(i));
  }
  private void verifyPrimeList(int[] list) {
    for (int i=0; i<list.length; i++)
      verifyPrime(list[i]);
  }
  private void verifyPrime(int n) {
    for (int factor=2; factor<n; factor++)
      assert(n%factor != 0);
  }
}
```

Quả là tai hoạ! Tôi cứ ngỡ là giải pháp nguyên thủy của tôi là thượng thặng. Chút gì đó tôi vẫn còn cảm thấy như vậy. Tôi cố phô trương tài năng của tôi 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, 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 tôi. 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.

* **Square root** hay **căn số** là cách gọi trước đây cho căn bậc 2, một cách gọi được dùng sau này ở VN (chú thích này dành cho những ai thắc mắc với 1 số thuật ngữ toán học xưa và nay ;))

** **max prime factor** hay **thừa số nguyên tố cực đại** (hoặc phân tố cực đại nguyên số) cũng là những thuật ngữ toán (quen dùng) trước đây. cực đại có thể dịch là tối đa nếu muốn (chú thích này một lần nữa dành cho những ai thắc mắc với 1 số thuật ngữ toán học xưa và nay ;))

<đón đọc phần kế tiếp>

The Crafsman 4. ATest Of Patient

Robert C. Martin

12 tháng 7, 2002

Nhật ký thân yêu,

Tối qua tôi ngồi tựa cửa sổ hàng giờ, nhìn các vì sao mờ dần trong 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 mảnh mã cho dù những mảnh mã này làm việc ngon lành.

Hôm nay với một bài tập mới, Jerry đến gặp tôi. Gã yêu cầu tôi viết một chương trình tính thừa số nguyên tố của số nguyên. Gã cho biết gã làm việc với tôi ngay từ đầu nên hai chúng tôi ngồi xuống và bắt đầu lập trình.

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. Thế nên tôi vớ lấy bàn đánh và bắt đầu viết mã. Khoảng nữa 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 < 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 < 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;
  }
}
```

Tôi kiểm tra chương trình bằng cách chạy nó với nhiều thông số khác nhau. Mọi thứ dường như ổn thoả. Chạy chương trình với giá trị thông số 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 mười lăm số hai.

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 "unit tests".

Jerry hỏi: "Mày làm gì vậy?"

"Chương trình chạy nên tôi đang viết các unit tests." Tôi đáp lại.

"Nếu chương trình đã chạy việc gì mày cần unit tests?" Gã hỏi tiếp.

Tôi không nghĩ đến điểm này. Tôi chỉ biết theo thông lệ cần phải viết unit tests. Tôi liều lĩnh đoán mò: "Để mà các lập trình viên khác biết được là chương trình đó chạy?"

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?"

Tôi đớ lưỡi không trả lời được nhưng gã ngăn tôi lại bằng một cái nhìn.

"OK", gã nói, "xoá hết những thứ mày đã làm đi. Tao chỉ cho mày cách tụi tao làm ở đây."

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, không tưởng tượng nổi.

Cuối cùng Jerry nói: "Xoá đi."

Tôi trả lời: "Nhưng chương trình đó chạy mà."

"Thì sao?" Jerry đáp lại.

Tôi bắt đầu nổi cáu. Tôi nói cứng: "Chương trình này chẳng có gì sai hết!"

"Thực vậy hở?" gã lầm bầm và vớ lấy bàn đánh, xoá hết mã nguồn của tôi.

Tôi điếng người. 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. 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 diễn nhanh qua trong đầu trong khi tôi nhìn gã chằm chặp.

"À, tao thấy mày nổi đoá rồi đó." Jerry nói một cách điềm tĩnh.

Tôi lắp bắp nhưng chẳng thốt được gì cho minh bạch.

"Này." Jerry nói, rõ ràng đang cố làm dịu tôi xuống. "Đừng có đeo cứng vào mã nguồn của mà 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.

Tôi buộc miệng: "Nhưng làm như thế thì quả là phí!"

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."

Gã nhìn tôi chừng một giây rồi tiếp tục. "Có bao giờ mày lỡ tay xoá cái gì đó mày đang làm chưa? cái gì đó mất của mày vài ngày làm việc đó?"

"Có một lần, ở trường". Tôi nói "Cái disk bị hỏng và hồ sơ lưu trữ cũ đến hai ngày."

Gã cau mày gật đầu biểu lộ sự thông hiểu rồi hỏi: "Mày mất bao lâu để tái tạo lại những cái đã bị mất?"

"Tôi nắm khá rõ những cái bị mất nên chỉ mất có nửa ngày để tái tạo."

"Ra thế mày chẳng thật sự mất khối lượng hai ngày làm việc."

Tôi chẳng màng gì đến cái logic của gã. Tôi không bắt bẻ được nhưng tôi không khoái cái logic đó. Chỉ đơn giản là tôi cảm thấy bị mất một khối lượng hai ngày làm việc!

Gã hỏi tiếp: "Mày có nhận thấy phần mã làm lại tốt hơn hay tệ hơn phần mã mày bị mất không?"

"Ồ, tốt hơn nhiều." Tôi nói, ngay lập tức hối tiếc là đã phát biểu như thế. "Lần thứ nhì tôi có thể dùng một cấu trúc tốt hơn nhiều."

Gã cười. "Thế thì cố thêm 25 phần trăm, mày đưa ra được một giải pháp tốt hơn."

Logic 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?"

Trả lời cho sự ngạc nhiên của tôi, gã gật đầu và nói: "Gần như là như vậy. Tao giả định chuyện vứt bỏ mã nguồn là một việc giá trị 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."

<đón xem phần kế tiếp>

The Crafsman 5. Baby Step

Robert C. Martin

24 tháng Bảy 2002

Jerry yêu cầu tôi viết một chương trình tạo ra các thừa số nguyên tố. Tôi viết xong, chương trình chạy ngon lành và sau đó gã xoá mất chương trình đó. Tôi khá bực nhưng Jerry bảo: "Đừng có quá đeo cứng vào mã nguồn của mình như thế."

Tôi chẳng khoái cái trò này nhưng tôi không có một chút luận cứ nào để chống chọi với gã. Tôi chỉ ngồi yên, bất đồng.

"OK", gã nói, "Mình làm lại từ đầu. Cách tụi tao làm ở đây là viết 'unit tests' trước".

Cái này đúng là kiểu nghiễn chuyện quái đản. Tôi nhanh trí phản ứng ngay: "Hở?"

"Để tao chỉ cho mày thấy." Gã nói. "Công tác của tụi mình là tạo ra một dãy các thừa số nguyên tố từ một số nguyên. Mày nghĩ được trường hợp test nào đơn giản nhất?"

"Trường hợp giá trị đầu tiên là 2. Và trong đó nó đưa về một dãy với chỉ một số 2."

"Đúng rồi." Gã nói. Và gã viết một cái unit test như sau:

```
public void testTwo() throws Exception {
  int factors[] = PrimeFactorizer.factor(2);
  assertEquals(1, factors.length);
  assertEquals(2, factors[0]);
}
```

Kế tiếp gã viết một đoạn mã rất đơn giản cho phép cái "test case" biên dịch.

```
public class PrimeFactorizer {
  public static int[] factor(int multiple) {
    return new int[0];
  }
}
```

Gã chạy thử cái test và nó báo lỗi: **"testTwo(TestPrimeFactors): expected: <1> but was: <0>"**.

Đến đây gã nhìn tôi và nói: "Làm cách gì đơn giản nhất để vượt qua cái test case đó."

Đây đúng là vô lý chồng chất. "Ý ông là sao?" Tôi hỏi. "Điều đơn giản nhất hẳn trả về một dãy với số 2 trong đó."

Gã trả lời với vẻ mặt nghiêm nghị: " OK, làm vậy đi."

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

"Ừa, đúng vậy." Gã đáp lời. "Nhưng khuấy nhộn lên một tí dùm tao đi."

Tôi thở dài bực dọc, phập phà một chút rồi viết:

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

Tôi chạy cái test và - tất nhiên nó ổn cả.

Tôi hỏi "Cái này chứng minh được điều gì vậy?"

"Nó chứng minh là mày có thể viết một cái hàm tìm ra thừa số nguyên tố của 2." Gã nói. "Nó cũng chứng minh là test đã ổn khi cái hàm trả lời đúng với số 2."

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

"Bây giờ, cái test case nào đơn giản nhất mình có thể đưa thêm vào?" gã hỏi tôi.

Tôi không kìm được, tôi chì chiết một cách diễu cợt: "Ôi, Jerry hay là mình nên thử với số 3?"

Và dẫu tôi hợm trước, không ngờ gã viết một cái "test case" cho số 3 thật:

```
public void testThree() throws Exception {
  int factors[] = PrimeFactorizer.factor(3);
  assertEquals(1, factors.length);
  assertEquals(3, factors[0]);
}
```

Chạy cái test này nó báo lỗi như đã đoán trước: **"testThree(TestPrimeFactors): expected: <3> but was: <2>"**.

"OK Alphonse, "Làm cách gì đơn giản nhất để cái test case này ổn."

Mất kiên nhẫn tôi vớ lấy bàn đánh và đánh vào như sau:

```
public static int[] factor(int multiple) {
  if (multiple == 2) return new int[] {2};
  else return new int[] {3};
}
```

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

Jerry nhìn tôi với một nụ cười bất thường. Gã nói: "OK, mấy cái test đó đạt. Tuy nhiên, nhìn ra không "chiến" lắm phải không?"

Gã là người bày cái trò ngớ ngẩn này và bây giờ gã đi hỏi tôi có chiến hay không? "Tôi cho rằng trọn bộ bài tập này khá nản đó." Tôi nói.

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

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

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

Các cái tests đều ổn cả và Jerry mỉm cười nhưng tôi vẫn không thể hình dung làm sao mấy cái trò này đưa đến vấn đề 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. Tuy nhiên tôi vẫn không ngạc nhiên mấy khi Jerry hỏi tôi: Bây giờ, cái test case nào đơn giản nhất mình có thể đưa thêm vào?"

"Rõ ràng là cho trường hợp số 4." Tôi nói một cáh thiếu kiên nhẫn và vớ lấy bàn đánh, tôi viết:

```
public void testFour() throws Exception {
```

```
  int factors[] = PrimeFactorizer.factor(4);
  assertEquals(2, factors.length);
  assertEquals(2, factors[0]);
  assertEquals(2, factors[1]);
}
```

Tôi nói "Tôi dự phỏng cái 'assert' thứ nhất sẽ hỏng vì chiều dài của dãy chỉ cho 1."

Quả vậy, khi chạy thử cái test nó tường trình: "**testFour(TestPrimeFactors) :expected <2> but was <1>"**.

Tôi hỏi: "Tôi đoán ông muốn tôi đưa ra thay đổi đơn giản nhất có thể làm cho các cái test đều ổn và tạo ra phương thức thừa số tổng quát hơn?"

Jerry gật đầu.

Tôi cố gắng phối hợp giải quyết cho cái test case trước mắt, lờ các test case tôi biết sẽ đụng đến sau. Cái trò này thật là ai oán nhưng Jerry muốn vậy. Kết quả như sau:

```
public class PrimeFactorizer {
  public static int[] factor(int multiple) {
    int currentFactor = 0;
    int factorRegister[] = new int[2];
    for (; (multiple % 2) == 0; multiple /= 2)
      factorRegister[currentFactor++] = 2;
    if (multiple != 1)
      factorRegister[currentFactor++] = multiple;
```

```
    int factors[] = new int[currentFactor];
    for (int i = 0; i < currentFactor; i++)
      factors[i] = factorRegister[i];
    return factors;
  }
}
```

Đoạn mã này qua hết các cái test nhưng nhìn khá lộn xộn. Jerry nhăn mặt như thể gã đánh được mùi hôi thối đâu đây. Gã nói: "Mình phải 'refactor' cái này trước khi đi tiếp."

"Hượm đã." Tôi phản đối. "Tôi đồng ý nó lộn xộn nhưng sao mình không làm cho nó chạy trước rồi 'refector' lại nếu có đủ thời gian?"

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

"OK." Tôi thở dài. "Thì dọn dẹp."

Thế rồi hai đứa tôi tách lượt phần mã này 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 < factorIndex; i++)
      factors[i] = factorRegister[i];
    return factors;
  }
}
```

Jerry tuyên bố: "Đến lúc cho cái test case kế tiếp." và gã chuyển bàn đánh đến tôi.

Tôi vẫn chưa thể nhận ra trò này đi đến đâu nhưng chỉ biết không cách gì thoát ra được. Một cách nhân nhượng tôi gõ cái test case như sau:

```
public void testFive() throws Exception {
  int factors[] = PrimeFactorizer.factor(5);
  assertEquals(1, factors.length);
  assertEquals(5, factors[0]);
}
```

"Thật là lý thú." Tôi nói trong khi nhìn chằm chặp vào cái 'bar' màu xanh, "nó chạy mà chẳng cần thay đổi gì hết."

"Đúng là lý thú". Jerry nối tiếp. "Hãy thử cái test case kế tiếp."

Lúc này tôi bị thu hút rõ. Tôi không dự tưởng cái test case chỉ chạy như vậy. Ngẫm nghĩ về vấn đề này, tôi cũng chưa hưởng ứng thực sự nhưng rõ ràng nó chạy. Tôi khá chắc cái test kết tiếp sẽ hỏng nên gõ đoạn test như sau 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 = "assertContains:" + n;
  for (int i = 0; i < factors.length; i++) {
    if (factors[i] == n)
      return;
  }
  fail(error);
}
```

"Ui! Cái test này cũng ổn luôn!" tôi rú lên.

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

"Vâng, tôi nghĩ vậy."

"vậy thì bỏ nó đi và đi thẳng tới 8, nó không qua được cái test đâu!"

Gã đúng. 8 phải hỏng vì dãy 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 = "assertContains(" + n + "," + f +")";
  Int int count = 0;
  for (int i = 0; i < factors.length; i++) {
    if (factors[i] == f)
      count++;
  }
  if (count != n)
    fail(error);
}
```

"Đúng là nhẹ nhõm!, nó hỏng rồi!"

"Ừa." Jerry đáp "vì bị quá giới hạn chiều dài của dãy. Mày có thể làm nó vượt qua được bằng cách gia tăng chiều dài của factorRegister nhưng cách này không tổng quát hơn được."

"Thì cứ thử xem sao rồi mình giải quyết vấn đề chiều dài của dãy sau."

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

"OK," tôi nói, "số cực đại của các thừa số mình có thể có là gì?"

"Tao nghĩ hình như là lôga 2 của một số hay sao đó." Jerry nói.

"Hẳng đã!" Tôi nói, "Có thể mình đang đi vòng vòng đây. Số lớn nhất mình có thể xử lý là mấy? không phải là 2 mũ 64 sao?"

Jerry đáp "Tao chắc là không thể hơn con số đó."

"OK, vậy thì thử tạo ra chiều dài của factorRegister là 100 đi. Nó lớn đủ để xử lý bất cứ số nào mình ném tới nó."

"Được thôi." Jerry nói "Một trăm số nguyên thì chẳng có gì phải lo."

Chúng tôi thử và các cái test vẫn chạy.

Tôi nhìn Jerry và nói: "test case kế tiếp của tôi đó nha. Chắc chắn nó sẽ hỏng."

Gã đáp "Thì thử đi."

Nên tôi gõ như sau:

```
public void testNine() throws Exception {
  int factors[] = PrimeFactorizer.factor(9);
  assertEquals(2, factors.length);
  assertContainsMany(factors, 2, 3);
}
```

"Trời, nó hỏng thật." Tôi nói. "Cho cái test qua được cũng đơn giản thôi. Tôi chỉ cần bỏ đi 2 như một số đặc biệt trong findPrimeFactors và dùng cả 2 và 3 cho thuật toán tổng quát." Thế là tôi điều chỉnh 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, đạt". Jerry nói. "Bây giờ xem thử cái test case tiếp theo nào hỏng?"

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

```
public void testSixteen() throws Exception {
  int factors[] = PrimeFactorizer.factor(16);
  assertEquals(4, factors.length);
  assertContainsMany(factors, 4, 2);
```

}

"Ui! Cái test này qua khỏi." Tôi nói "Làm sao nó qua khỏi được cà?"

"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 hoặc là 8, hoặc là 4!"

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

"Đúng thế." Jerry nói. "Đó là lý do tại sao tao xoá nó."

"Vậy thì xong? Mình hoàn thành rồi phải không?"

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

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

"À, kiểu chơi ôm đồm. OK, thử đi."

```
public void testThousand() throws Exception {
  int factors[] = PrimeFactorizer.factor(1000);
  assertEquals(6, factors.length);
  assertContainsMany(factors, 3, 2);
  assertContainsMany(factors, 3, 5);
```

}

"Nó chạy luôn! OK, hay là..."

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

Điều làm tôi kinh ngạc và vẫn còn làm tôi kinh ngạc là sau mỗi cái test case chúng tôi lại tuồn ra một giải pháp tốt hơn. Nếu không rấn lên mỗi lần một test case thì tôi không nghĩ sẽ bao giờ dính vào lối khai triển đơn giản này. Tôi không biết chuyện gì sẽ xảy ra với những projects lớn hơn nữa?

Hôm nay tôi đã học được đôi điều.

<đón xem phần kế tiếp>

The Crafsman 6. Socket Service

Robert C. Martin

16 tháng Chín 2002

Sự kiện ngày hôm qua làm tôi lả người. Jerry và tôi giải quyết xong vấn đề tạo thừa số nguyên tố bằng cách tuồn qua mỗi lần một test case tí hon. Đây là một cách giải quyết vấn đề kỳ lạ nhất mà tôi từng thấy nhưng nó lại làm việc ngon lành hơn giải pháp nguyên thủy của tôi.

Tôi lẩn quẩn vô định hướng trong các hành lang, ngẫm nghĩ đến chuyện này mãi. Tôi chẳng còn nhớ đến bữa tối hay ở đâu nữa. Tôi ngủ thiếp đi sớm hơn ngày thường và chiêm bao về những phân đoạn của mấy cái test bé nhỏ kia.

Sáng nay khi tôi trình diện Jerry, gã nói:

"Chào Alphonse. Mày đã sẵn sàng cho một chương trình thật chưa?"

"Ông thừa biết như thế! Thích quá, vâng, tôi sẵn sàng! Tôi quá mệt mấy cái trò thử nghiệm này lắm rồi."

"Tốt lắm! Tụi mình có một chương trình gọi là SMC dùng để biên dịch trạng thái ngữ pháp hữu hạn của máy vào môi trường Java. Ông C muốn tụi mình biến chương trình ấy thành một dịch vụ trên mạng."

"Ý ông là sao?", tôi hỏi.

Jerry xoay qua bản phác thảo rồi bắt đầu cùng một lúc giảng giải và minh hoạ.

"Mình sẽ viết hai chương trình. Một cái gọi là SMCR Client và cái kia gọi là SMCR Server. Người dùng muốn biên dịch trạng thái ngữ pháp hữu hạn của máy sẽ dùng tên của hồ sơ để gọi SMCR Client. SMCR Client sẽ gởi hồ sơ đó đến một máy đặc biệt nơi SMCR Server đang hoạt động. SMCR Server sẽ chạy phần biên dịch SMC và gởi kết quả biên dịch về SMCR Client. SMCR Client sẽ viết các dữ kiện này vào thư mục của người dùng. Đối với người dùng, cơ chế này không khác gì họ đang dùng SMC trực tiếp."

"OK, tôi nghĩ là tôi hiểu vấn đề." Tôi nói. "Nghe khá đơn giản."

"Nó khá đơn giản thật." Jerry đáp. "Nhưng đụng đến sockets lúc nào cũng thú vị hơn một chút."

Chúng tôi ngồi xuống máy và, như thường lệ, chuẩn bị tư thế viết cái unit test đầu tiên. Jerry suy nghĩ một lúc rồi trở lại bản phác thảo và phác hoạ ra một biểu đồ như sau:

"Đây là ý nghĩ của tao về SMCR Server." Gã nói. "Chúng ta sẽ đặt mã quản lý socket vào class **SocketService**. Class này sẽ đón và quản trị các truy cập từ bên ngoài vào. Khi **serve(port)** được gọi, nó sẽ tạo một dịch vụ socket với port đã ấn định và bắt đầu tiếp nhận truy cập. Bất cứ khi nào có một truy cập xảy ra nó sẽ tạo một thread mới và chuyển giao nhiệm vụ điều tác sang method **serve(socket)** thuộc interface **SocketServer**. Với cách đó, mình tách rời mã quản trị socket ra khỏi phần mã mình muốn dùng để thao tác các dịch vụ khác."

Không nắm được lối khai triển này hiệu quả hay không, tôi chỉ gật đầu. Rõ ràng gã có lý do để nghĩ như thế. Tôi chỉ theo đuôi mà thôi.

Kế tiếp gã viết cái test như sau:

```
public void testOneConnection() throws Exception {

  SocketService ss = new SocketService();

  ss.serve(999);
```

```
  connect(999);

  ss.close();

  assertEquals(1, ss.connections());

}
```

"Chuyện tao làm ở đây có cái tên ***Intentional Programming*** (lập trình có chủ định). Jerry nói. "Tao gọi cái đoạn mã trong lúc nó chưa tồn tại. Làm như thế để diễn đạt chủ ý của mình về phương diện mã nguồn sẽ ra sao, làm việc như thế nào."

"OK." Tôi đáp. "Ông tạo cái **SocketService** rồi ông chỉ định nó tiếp nhận các truy cập trên port 999. Kế tiếp có vẻ như ông truy cập vào dịch vụ mới vừa được tạo ra trên port 999. Cuối cùng ông đóng **SocketService** và 'assert' rằng nó có một truy cập."

"Đúng như thế." Jerry xác nhận.

"Nhưng làm sao ông biết **SocketService** sẽ cần các **connections** method?"

"Ô, có lẽ nó không cần. Tao chỉ đặt nó ở đó để có thể test nó."

"Như vậy không phí sao?" Tôi dạm hỏi.

Jerry nghiêm khắc nhìn tôi và trả lời: "Không có gì làm cho một cái test được dễ dàng lại là phung phí cả Alphonse. Tụi tao thường thêm các methods và các classes đơn giản để tạo điều kiện test các classes dễ dàng hơn."

Tôi không khoái cái **connnections()** method nhưng cứ làm thinh.

Chúng tôi chỉ viết vừa đủ phần contructor **SocketService** và các methods **serve**, **close** và **connect** để có thể biên dịch trọn bộ. Các functions này đều trống nên khi chúng tôi chạy thử, cái test bị hỏng như dự đoán.

Kế tiếp Jerry viết method connect như một phần của cái class cho test case

```
private void connect(int port) {
  try {
    Socket s = new Socket("localhost", port);
    s.close();
  }
  catch (IOException e) {
    fail("could not connect");
  }
}
```

Chạy test này báo lỗi như sau: "**testOneConnection: could not connect"**

Tôi nói: "Nó hỏng vì không thể tìm ra port 999 ở đâu hết, đúng không?"

"Đúng vậy!" Jerry trả lời. "Nhưng chuyện đó dễ thôi. Đây, sao mày không sửa nó đi?"

Trước giờ tôi chưa bao giờ viết mã cho socket nên không biết phải làm tiếp những gì. Jerry chỉ tôi đến phần **ServerSocket** trong Javadocs. Các ví dụ ở đây xem ra rất đơn giản nên tôi ngoáy thêm trong các methods của **SocketService** như sau:

```
public class SocketService {
```

```
  private ServerSocket serverSocket = null;

  public void serve(int port) throws Exception {
    serverSocket = new ServerSocket(port);
  }

  public void close() throws Exception {
    serverSocket.close();
  }

  public int connections() {
    return 0;
  }
}
```

Chạy phần mã này nó báo: **"testOneConnection: expected: <1> but was: <0>"**

"À ha!" Tôi nói: "Nó tìm ra port 999. Quá đã! nhưng mình cần đếm số lần truy cập!"

Nên tôi đổi **SocketService** class như sau:

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class SocketService {
  private ServerSocket serverSocket = null;
  private int connections = 0;

  public void serve(int port) throws Exception {
    serverSocket = new ServerSocket(port);
    try {
      Socket s = serverSocket.accept();
      s.close();
      connections++;
    }
    catch (IOException e) {
    }
  }

  public void close() throws Exception {
    serverSocket.close();
  }

  public int connections() {
    return connections;
```

```
  }

}
```

Nhưng đoạn mã này không chạy, nó cũng chẳng báo lỗi. Khi tôi chạy phần test, nó bị treo.

"Chuyện gì đây cà?" Tôi thắc mắc.

Jerry mỉm cười. "Thử xem mày có thể mò ra không Alphonse. Dò thử đi."

"OK, xem thử. Cái chương trình test gọi **serve** để tạo ra socket và tiếp tục gọi **accept**. Ồ! **accept** không trả về cho đến khi nó có được một truy cập, và vì **serve** không hề trả lại nên mình không hề có cơ hội gọi **connect**."

Jerry gật đầu. "Vậy thì mày định sửa nó thế nào Alphonse?"

Tôi nghĩ ngợi một chút. Tôi cần gọi function **connect** sau khi gọi **accept** nhưng khi mình gọi **accept** nó không trả về cho đến khi mình gọi **connect**. Nhìn qua thì có vẻ không thể được.

"Không phải là không được đâu Alphonse." Jerry cất tiếng. "Mày chỉ cần tạo ra một cái thread."

Tôi lại ngẫm nghĩ thêm một chút nữa. Đúng rồi, tôi có thể đặt phần gọi cho việc tiếp nhận truy cập trong một thread khác rồi mới bắt lấy thread đó và gọi bước truy cập.

Tôi nói: "Tôi biết lý do tại sao ông nói tạo mã nguồn cho socket thú vị hơn một chút rồi đó." và tôi thay đổi đoạn mã như sau:

```
private Thread serverThread = null;


public void serve(int port) throws Exception {
  serverSocket = new ServerSocket(port);
  serverThread = new Thread(
    new Runnable() {
      public void run() {
        try {
          Socket s = serverSocket.accept();
          s.close();
          connections++;
        }
        catch (IOException e) {
        }
      }
    }
  );
```

```
  serverThread.start();

}
```

"Sử dụng cái anonymous inner class hay lắm đó Alphonse." Jerry nói.

"Cám ơn." Tôi cảm thấy sương sướng khi được gã khen. "Nhưng e nó tạo một chùm đuôi khỉ ở cuối cái function."

"Mình refactor nó sau, đầu tiên cứ chạy cái test cái đã."

Cái test chạy ổn nhưng Jerry có vẻ đăm chiêu, như thể gã vừa bị ai nói dối.

"Chạy cái test lần nữa xem Alphonse."

Tôi vui vẻ nhấn nút run và cái test lại chạy ngon lành.

"Lần nữa." Gã nói.

Tôi nhìn gã một giây xem thử gã có đùa không. Rõ ràng gã không đùa. Mắt gã dán chặt trên màn hình như thể gã đang săn lùng "dribin". Thế nên tôi nhấn nút một lần nữa và thấy: **"testOneConnection: expected:<1> but was:<0>"**

"Hẵng đã!" tôi rú lên. "Không thể nào!"

"Ồ, có thể chớ." Jerry nói. "Tao đang đợi nó xảy ra."

Tôi nhấn nút liên tục. Trong mười lần có đến ba lần hỏng. Không biết tôi có loạn trí không? làm sao chương trình lại giở trò như vậy?

"Làm sao ông biết được vậy Jerry? Ông có liên hệ gì đến sấm truyền Aldebran hở?"

"Không, tao có viết loại mã này trước đây nên biết đôi điều cần dự phòng. Mày có thể minh giải chuyện gì xảy ra không? Suy nghĩ cho thấu đáo và kỹ càng đó."

Đúng là đau đầu nhưng tôi bắt đầu ráp từng phần lại với nhau. Tôi đến bản phác thảo và vẽ ra:

Khi đã minh giải xong, tôi tường trình sự vụ cho Jerry. "**TestSocketServer** gởi thông điệp **serve(999)** đến **SocketService**. **SocketService** tạo **ServerSocket** và **serverThread** rồi trả về. Sau đó **TestSocketServer** gọi **connect** phân đoạn đã tạo nên client socket. Hai sockets này hẳn đã tìm thấy nhau bởi vì chúng ta không nhận được lỗi 'could not connect'. **ServerSocket** hẳn đã tiếp nhận truy cập nhưng có lẽ **serverThread** chưa có cơ hội để chạy. Và trong khi **serverThread** bị cản, function **connect** đóng client socket lại. Kế tiếp **TestSocketServer** gởi thông điệp đóng cửa đến **SocketService** và phân đoạn này đóng **serverSocket**. Khi **serverThread** có cơ hội gọi function **accept** thì server socket đã đóng mất."

"Tao nghĩ mày đúng đó." Jerry nói. "Hai biến cố - tiếp nhận và đóng - thiếu đồng bộ và hệ thống này dễ hỏng với các trình tự xảy ra. Cái này mình gọi là trường hợp dồn đuổi (race condition). Chúng ta phải bảo đảm thắng cuộc đuổi chạy này."

Chúng tôi quyết định thử nghiệm giả thuyết của tôi bằng cách đưa vào các print statement trong khối 'catch' sau khi **accept** được gọi. Hẳn vậy, trong mười lần test, chúng tôi thấy thông điệp này ba lần.

Jerry hỏi tôi: "Thế thì làm sao mình cho unit test chạy đây?"

"Theo tôi nghĩ, dường như cái test không thể chỉ mở client socket rồi đóng lại ngay lập tức." Tôi đáp "Nó cần phải đợi bước tiếp nhận."

Gã nói: "Mình có thể đợi 100ms trước khi đóng client socket."

"Ừa, chắc là được nhưng hơi ẹ." Tôi trả lời.

"Hãy xem thử mình làm cho nó chạy được hay không cái đã rồi tính chuyện refector sau."

Nên tôi thay đổi method **connect** như sau:

```
private void connect(int port) {
  try {
    Socket s = new Socket("localhost", port);
    try {
      Thread.sleep(100);
```

```
    }
    catch (InterruptedException e) {
    }
    s.close();
  }
  catch (IOException e) {
    fail("could not connect");
  }
}
```

Phần thay đổi cho kết quả test 10 trên 10.

"Gớm thật." Tôi nói. "Khi mình đối phó với nhiều threads thì phải dè chừng trường hợp dồn đuổi (race condition). Nhấn nút test nhiều lần là một thói quen tốt nên tập."

"Hên là mình khám phá ra nó trong mấy cái test case." Tôi nói. "Không thì khó mà kiếm ra nó sau khi hệ thống đã chạy."

Jerry chỉ gật đầu.

<đón xem phần kế tiếp>

The Crafsman 7. Socket Service 2

Robert C. Martin

Ngày 15 tháng 10 2002

Lần trước Alphonse và Jerry khởi đầu trên một framework java đơn giản hỗ trợ dịch vụ socket. Test case thứ nhất của họ vạch ra trường hợp dồn đuổi (race condition) mà họ đã giải quyết ổn thoả. Chuỗi unit test hiện tại được dẫn ở **Mã dẫn 1** và mã nguồn chính ở **Mã dẫn 2**.

Mã dẫn 1

```
import junit.framework.TestCase;

import junit.swingui.TestRunner;

import java.io.IOException;

import java.net.Socket;


public class TestSocketServer extends TestCase {

 public static void main(String[] args) {

   TestRunner.main(new String[]{"TestSocketServer"});

 }


 public TestSocketServer(String name) {

  super(name);

 }


 public void testOneConnection() throws Exception {

  SocketService ss = new SocketService();

  ss.serve(999);

  connect(999);
```

```
    ss.close();
    assertEquals(1, ss.connections());
  }

  private void connect(int port) {
    try {
      Socket s = new Socket("localhost", port);
      try {
        Thread.sleep(100);
      }
      catch (InterruptedException e) {
      }
      s.close();
    }
    catch (IOException e) {
      fail("could not connect");
    }
  }
}
```

Mã dẫn 2

```
import java.io.IOException;
import java.net.*;

public class SocketService {
 private ServerSocket serverSocket = null;
 private int connections = 0;
 private Thread serverThread = null;

 public void serve(int port) throws Exception {
  serverSocket = new ServerSocket(port);
  serverThread = new Thread(
   new Runnable() {
    public void run() {
     try {
      Socket s = serverSocket.accept();
      s.close();
      connections++;
     }
     catch (IOException e) {
```

```
        }
      }
    }
   );
   serverThread.start();
  }

  public void close() throws Exception {
   serverSocket.close();
  }

  public int connections() {
   return connections;
  }
}
```

Sau giờ giải lao, chúng tôi trở lại và sẵn sàng tiếp túc với **SocketService**.

"Chúng ta đã chứng minh được mình có thể truy cập một lần. Vậy hãy thử truy cập nhiều lần xem sao." Jerry nói.

"Nghe được lắm." Tôi trả lời. Sau đó tôi viết cái test case như sau:

```
public void testManyConnections() throws Exception {
  SocketService ss = new SocketService();
  ss.serve(999);
```

```
  for (int i = 0; i < 10; i++)
    connect(999);
  ss.close();
  assertEquals(10, ss.connections());
}
```

"OK, cái test này hỏng." Tôi nói.

"Y như ước đoán". Jerry đáp. "Cái SocketService chỉ gọi method accept một lần. Chúng ta cần đặt bước gọi đó vào một vòng lặp."

"Khi nào vòng lặp đó chấm dứt?" Tôi hỏi.

Jerry nghĩ ngợi 1 tí và nói: "Khi chúng ta gọi method close của SocketService."

"Như thế này chăng?" Và tôi hiệu đính như sau:

```
public class SocketService {
  private ServerSocket serverSocket = null;
  private int connections = 0;
  private Thread serverThread = null;
  private boolean running = false;

  public void serve(int port) throws Exception {
    serverSocket = new ServerSocket(port);
    serverThread = new Thread(
```

```
      new Runnable() {
        public void run() {
          running = true;
          while (running) {
            try {
              Socket s = serverSocket.accept();
              s.close();
              connections++;
            }
            catch (IOException e) {
            }
          }
        }
      }
    );
    serverThread.start();
  }

  public void close() throws Exception {
    running = false;
    serverSocket.close();
  }
}
```

Tôi chạy cái test và cả hai đều đạt.

"Tốt." Tôi nói. "Bây giờ chúng ta có thể truy cập bao nhiêu tùy thích. Không may cái **SocketService** chẳng làm gì nhiều khi mình truy cập đến nó. Nó chỉ đóng lại mà thôi."

"Ừa, đổi nó đi." Jerry nói. "Mình hãy buộc **SocketService** gởi thông điệp "Hello" mỗi khi chúng ta truy cập đến nó."

Tôi không cần chuyện này cho lắm. Tôi nói: "Tại sao mình làm bẩn cái **SocketService** bằng thông điệp "Hello" chỉ để thoả mãn cái test của mình? **SocketService** có thể gởi thông điệp thì tốt nhưng mình không muốn thông điệp này là một phần của mã nguồn **SocketService**!"

"Đúng thế!" Jerry đồng ý. "Mình muốn thông điệp được chỉ định và xác thực do cái test."

"Mình làm sao đây?" Tôi hỏi.

Jerry mỉm cười đáp: "Chúng ta dùng cái **Mock Object pattern**. Nói một cách ngắn gọn, mình tạo ra một cái interface từ đó **SocketService** sẽ thao tác sau khi nhận một truy cập. Chúng ta sẽ có cái test ứng dụng cái interface đó dùng để gởi thông điệp "Hello". Sau đó, mình sẽ có cái test dùng để đọc thông điệp từ socket của client và xác thực thông tin được gởi đi một cách đúng đắn."

Tôi chẳng biết **Mock Object** pattern là gì cả và thành phần interface của gã làm tôi bối rối. "Ông chỉ cho tôi được không?" Tôi hỏi.

Thế rồi Jerry vớ lấy bàn đánh và bắt đầu gõ.

"Đầu tiên chúng ta viết cái test."

```
public void testSendMessage() throws Exception {

  SocketService ss = new SocketService();

  ss.serve(999, new HelloServer());

  Socket s = new Socket("localhost", 999);

  InputStream is = s.getInputStream();

  InputStreamReader isr = new InputStreamReader(is);

  BufferedReader br = new BufferedReader(isr);

  String answer = br.readLine();

  s.close();

  assertEquals("Hello", answer);

}
```

Tôi kiểm tra đoạn mã này cẩn thận. "OK, ông tạo ra cái gọi là **HelloServer** và đưa nó vào trong method **serve**. Cái này sẽ làm hỏng hết các cái test khác!"

"Hay lắm!" Jerry thốt lên. "Điều đó có nghĩa là chúng ta cần refactor những test khác trước khi tiếp tục."

"Nhưng các dịch vụ trong hai cái test kia chẳng làm gì hết." Tôi ý kiến.

"Tất nhiên là chúng làm việc - chúng đếm số truy cập! Mày có nhớ là mày ghét mấy cái biến số truy cập đến thế nào không, và nó chỉ là phần phụ mà thôi? Bây giờ mình sẽ dẹp chúng đi."

"Mình sắp sửa làm thế à?"

"Xem đây." Jerry cười rộ. "Đầu tiên chúng ta đổi hai cái test và thêm biến số **connections** vào test case."

```
public void testOneConnection() throws Exception {
  ss.serve(999, connectionCounter);
  connect(999);
  assertEquals(1, connections);
}

public void testManyConnections() throws Exception {
  ss.serve(999, connectionCounter);
  for (int i=0; i<10; i++)
   connect(999);
  assertEquals(10, connections);
}
```

"Kế tiếp mình tạo cái interface."

```
import java.net.Socket;
public interface SocketServer {
  public void serve(Socket s);
}
```

"Sau đó chúng ta tạo biến số **connectionCounter** và khởi động nó trong constructor của **TestSocketServer** bằng một anonymous inner class để nó tăng cấp biến số **connections**.

```
public class TestSocketServer extends TestCase {
  private int connections = 0;
  private SocketServer connectionCounter;

  public static void main(String[] args) {
    TestRunner.main(new String[]{"TestSocketServer"});
  }

  public TestSocketServer(String name) {
    super(name);
    connectionCounter = new SocketServer() {
      public void serve(Socket s) {
      connections++;
    }
  };
}
...
```

"Cuối cùng, chúng ta cho nó biên dịch trọn bộ bằng cách thêm đối số phụ vào method **serve** của **SocketService** và biến cái test mới thành phần chú giải (để nó khỏi chạy)."

```
public void serve(int port, SocketServer server) throws Exception {
  ...
}
```

"OK, tôi biết ý ông rồi." Tôi nói. "Hai cái test cũ lúc này hẳn phải hỏng bởi lẽ **SocketService** không bao giờ gây ra method serve từ đối số **SocketServer** của nó."

Tất nhiên các cái test đã hỏng vì chính lý do ấy.

Tôi biết phải làm gì kế tiếp. Tôi vớ lấy bàn đánh và thay đổi như sau:

```
public class SocketService {
 private ServerSocket serverSocket = null;
 private int connections = 0;
 private Thread serverThread = null;
 private boolean running = false;
 private SocketServer itsServer;

 public void serve(int port, SocketServer server) throws Exception {
  itsServer = server;
  serverSocket = new ServerSocket(port);
  serverThread = new Thread(
   new Runnable() {
    public void run() {
     running = true;
     while (running) {
      try {
       Socket s = serverSocket.accept();
       itsServer.serve(s);
```

```
            s.close();
            connections++;
          } catch (IOException e) {
        }
      }
    }
  }
 );
 serverThread.start();
}
...
```

Đoạn mã này làm các test chạy được.

"Hay lắm!" Jerry nói. "Bây giờ chúng ta phải làm cho cái test mới chạy."

Thế nên tôi tháo bỏ phần chú giải cho đoạn test và biên dịch nó. Nó "la làng" trong phần **HelloServer**.

"Ô, đúng rồi. Mình phải thực thi cái **HelloServer**. Nó sẽ phun ra chữ "hello" từ socket, phải không?"

"Đúng thế." Jerry xác nhận.

Thế rồi tôi viết cái class mới trong hồ sơ **TestSocketServer.java** như sau

```
class HelloServer implements SocketServer {

  public void serve(Socket s) {

    try {

      PrintStream ps = new PrintStream(s.getOutputStream());

      ps.println("Hello");

    }

    catch (IOException e) {

    }

  }

}
```

Các test đều ổn.

"Cũng dễ thôi." Jerry nói.

"Ừa. Phần pattern Mock Oject khá hữu dụng. Nó cho phép ta duy trì các mã dùng để test trong kế hoạch test. **SocketService** không biết gì cả."

"Còn hữu dụng hơn thế." Jerry trả lời. "Các servers thật cũng sẽ ứng dụng interface **SocketServer**."

"Tôi biết." Tôi trả lời. "Thật lý thú khi thấy từ nhu cầu tạo ra một unit test đưa mình đến chỗ tạo ra một đồ hình hữu dụng một cách tổng quát."

"Điều này thường xảy ra mà." Jerry nói. "Tests là người dùng đó. Nhu cầu dùng tests thường trùng hợp với nhu cầu của người dùng thật sự."

"Nhưng tại sao lại gọi nó là Mock Object?"

"Hãy nghĩ trên phương diện thế này. **HelloServer** dùng để thay thế cho, hoặc là một bản nháp, của một server thật. Cái pattern này cho phép chúng ta thay thế bản nháp của chuyện test vào mã nguồn ứng dụng thật sự."

"À ra vậy." Tôi đáp. "Thôi thì bây giờ mình nên dọn dẹp phần mã này và xoá bỏ cái biến số truy cập vô dụng kia vậy."

"Đồng ý."

Thế rồi chúng tôi dọn dẹp thêm một chút nữa và nghỉ giải lao. Kết quả của **SocketService** như sau:

```
import java.io.IOException;

import java.net.*;


public class SocketService {

  private ServerSocket serverSocket = null;

  private Thread serverThread = null;

  private boolean running = false;

  private SocketServer itsServer;
```

```
public void serve(int port, SocketServer server) throws Exception {
  itsServer = server;
  serverSocket = new ServerSocket(port);
  serverThread = makeServerThread();
  serverThread.start();
}

private Thread makeServerThread() {
  return new Thread (
    new Runnable() {
      public void run() {
        running = true;
        while (running) {
          acceptAndServeConnection();
        }
      }
    }
  );
}

private void acceptAndServeConnection() {
  try {
    Socket s = serverSocket.accept();
    itsServer.serve(s);
    s.close();
```

```
    }
    catch (IOException e) {
    }
  }

  public void close() throws Exception {
    running = false;
    serverSocket.close();
  }
}
```

<đón xem phần kế tiếp>

The Crafsman 8. Testing in Sync

"Testing in Synch", tay học việc của chúng ta học được một điều: các tests có mục đích phục vụ lớn hơn là chỉ đơn thuần chứng minh là mã nguồn chạy được: "tests" là một dạng tài liệu thực hành và giáo dục.

Robert C. Martin

Thang máy tầng 17 lại hỏng nên tôi phải dùng trụ tuột. Trong lúc tụt xuống, tôi bắt đầu ngẫm nghĩ đến chuyện đáng ghi nhận là dùng tests như một thứ đồ nghề thiết kế. Chìm đắm trong suy nghĩ, tôi hơi vô ý nên va cùi chỏ vào trụ thang với sức dội Coriolis [*]. Khi tôi gặp Jerry trong phòng thí nghiệm nó vẫn còn đau nhói.

"Mày sẵn sàng thử cái test dùng để gởi thông điệp 'hello' xuyên qua sockets chưa?" gã hỏi.

"Hiển nhiên rồi", tôi đáp.

Chúng tôi bỏ phần phụ chú (comment) của method TestSendMessage.

```
public void testSendMessage() throws Exception {
  SocketService ss = new SocketService();
  ss.serve(999, new HelloServer());
  Socket s = new Socket("localhost", 999);
  InputStream is = s.getInputStream();
  InputStreamReader isr = new InputStreamReader(is);
  BufferedReader br = new BufferedReader(isr);
  String answer = br.readLine();
  s.close();
  assertEquals("Hello", answer);
}
```

"Như dự đoán, đoạn này không biên dịch được" Jerry phát biểu. "Mình cần phải viết cái HelloServer."

"Tôi nghĩ tôi biết phải làm gì," tôi trả lời. "HelloServer là cái class thừa hưởng từ SocketServer và ứng dụng method server() để gởi thông điệp 'hello' qua socket."

Tôi vớ lấy bàn đánh và điều chỉnh nhóm TestSocketServer.java như sau:

```
class HelloServer implements SocketServer {
  public void serve(Socket s) {
    try {
      OutputStream os = s.getOutputStream();
      PrintStream ps = new PrintStream(os);
      ps.println("Hello");
    }
    catch (IOException e) {
    }
  }
}
```

Đoạn này biên dịch được và các tests đều đạt ngay lần đầu.

"Ngon lành," Jerry nói. "Bây giờ chúng ta có thể gởi một thông điệp qua socket."

Tôi biết Jerry bắt đầu nghĩ đến chuyện refactoring và tôi muốn qua mặt gã. Xem xét đoạn mã kỹ lưỡng, tôi nhớ gã có đề cập đến vấn đề trùng lặp.

"Có một số mã trùng lặp trong các phần unit tests," tôi nói. "Trong mỗi cái test mình tạo và đóng cái SocketService. Chúng ta nên bỏ nó đi."

"Tinh mắt lắm!" Jerry phán. "Hãy dời nó vào các function Setup và Teardown." Gã tóm lấy bàn đánh và thay đổi như sau:

```
private SocketService ss;
```

```
public void setUp() throws Exception {

  ss = new SocketService();

}


public void tearDown() throws Exception {

  ss.close();

}
```

Sau đó gã bỏ trọn bộ các dòng ss = newSocketService(); and ss.close(); trong ba cái tests.

"Xem được hơn đó," tôi nói. "Hãy thử xem mình có thể gởi một thông điệp ngược lại không."

"Tao cũng nghĩ y như vậy," Jerry trả lời. "Và tao có một cách làm chuyện đó."

Gã bắt đầu đánh một test case mới:

```
public void testReceiveMessage() throws Exception {

  ss.serve(999, new EchoService());

  Socket s = new Socket("localhost", 999);

  InputStream is = s.getInputStream();

  InputStreamReader isr = new InputStreamReader(is);

  BufferedReader br = new BufferedReader(isr);

  OutputStream os = s.getOutputStream();

  PrintStream ps = new PrintStream(os);

  ps.println("MyMessage");
```

```
  String answer = br.readLine();

  s.close();

  assertEquals("MyMessage", answer);

}
```

"Eo ôi! Tởm thế," tôi cằn nhằn.

"Ừa, đúng thật," Jerry thú nhận. "Hãy làm cho nó chạy cái đã rồi mình dọn dẹp nó sau. Chúng ta không muốn mớ lộn xộn đó ở đây lâu! Mày biết tao định làm gì phải không?"

"Vâng," Tôi trả lời. "EchoService sẽ nhận một thông điệp từ socket và gởi ngược lại ngay. Bởi thế, đoạn test của ông chỉ gởi MyMessage; rồi đọc nó lại."

"Đúng rồi. muốn thử ngoáy phần EchoService không?" "Tất nhiên," tôi nói một cách hăm hở.

```
class EchoService implements SocketServer {

  public void serve(Socket s) {
```

```
    try {
      InputStream is = s.getInputStream();
      InputStreamReader isr = new InputStreamReader(is);
      BufferedReader br = new BufferedReader(isr);
      OutputStream os = s.getOutputStream();
      PrintStream ps = new PrintStream(os);
      String token = br.readLine();
      ps.println(token);
    }
    catch (IOException e) {
    }
  }
}
```

"Oái," tôi nói. "Lại thêm một mớ mã xấu xí. Mình cứ tạo các objects PrintStream và BufferedReader từ socket. Chúng ta cần phải dọn dẹp mới được."

"Mình sẽ làm chuyện đó ngay sau khi mấy cái test chạy ngon lành," Jerry đáp, trong khi nhìn tôi có vẻ kỳ vọng.

"Oh!" tôi rú lên. "Tôi quên chạy cái test." Xấu hổ, tôi nhấn nút test và theo dõi nó chạy. "Cũng không khó lắm," tôi nói. "Bây giờ hãy vứt phần mã xấu xí ấy đi." Tôi rút nhiều functions ra khỏi EchoService.

```
class EchoService implements SocketServer {
  public void serve(Socket s) {
    try {
```

```
    BufferedReader br = getBufferedReader(s);
    PrintStream ps = getPrintStream(s);
    String token = br.readLine();
    ps.println(token);
   }
   catch (IOException e) {
   }
  }
  private PrintStream getPrintStream(Socket s)  throws IOException {
   OutputStream os = s.getOutputStream();
   PrintStream ps =  new PrintStream(os);
   return ps;
  }
  private BufferedReader getBufferedReader(Socket s)  throws IOException {
   InputStream is = s.getInputStream();
   InputStreamReader isr =  new InputStreamReader(is);
   BufferedReader br =  new BufferedReader(isr);
   return br;
  }
}
```

"Đoạn này cải tiến method EchoService," Jerry nói, "nhưng nó khá rối cái class. Hơn nữa, nó không giúp gì cho function testRecieveMessage, đó cũng là một điểm không đẹp. Thử nghĩ getBufferedReader và getPrintStream có nằm đúng chỗ không?"

"Đây sẽ là vấn đề lặp lại," tôi nói. "Ai muốn dùng SocketService phải sẽ chuyển socket thành BufferedReader và PrintStream."

"Chính là câu trả lời!" Jerry đáp lại. "Các methods getBufferedReader và getPrintStream quả thực thuộc về SocketService."

Tôi dời hai functions vào class SocketService và thay đổi EchoService theo đó.

```
public class SocketService {
 [...]
 public static PrintStream getPrintStream(Socket s) throws IOException {
  OutputStream os = s.getOutputStream();
  PrintStream ps = new PrintStream(os);
  return ps;
 }
 public static BufferedReader getBufferedReader(Socket s) throws IOException{
  InputStream is = s.getInputStream();
  InputStreamReader isr = new InputStreamReader(is);
  BufferedReader br = new BufferedReader(isr);
  return br;
 }
}

class EchoService implements SocketServer {
 public void serve(Socket s) {
  try {
   BufferedReader br = SocketService.getBufferedReader(s);
   PrintStream ps = SocketService.getPrintStream(s);
```

```
      String token = br.readLine();
      ps.println(token);
    }
    catch (IOException e) {
    }
  }
}
```

Các tests đều chạy. Giữ vững tình hình, tôi nói: "bây giờ tôi hắn có thể sửa đổi method testReceiveMessage luôn." Trong lúc Jerry quan sát, t ôi thay đổi phần này như sau:

```
public void testReceiveMessage() throws Exception {
  ss.serve(999, new EchoService());
  Socket s = new Socket("localhost", 999);
  BufferedReader br = SocketService.getBufferedReader(s);
  PrintStream ps = SocketService.getPrintStream(s);
  ps.println("MyMessage");
  String answer = br.readLine();
  s.close();
  assertEquals("MyMessage", answer);
}
```

"Ừa, coi được hơn đó," Jerry nói.

"Không chỉ như vậy mà các tests đều đạt," tôi gáy lên. Thế rồi tôi nhận thấy thêm một điều. "Ui, có thêm một cái nữa trong testSendMessage." Tôi sửa cái đó luôn.

```
public void testSendMessage() throws Exception {
  ss.serve(999, new HelloServer());
  Socket s = new Socket("localhost", 999);
  BufferedReader br = SocketService.getBufferedReader(s);
  String answer = br.readLine();
  assertEquals("Hello", answer);
}
```

Các tests vẫn chạy. Tôi ngồi tẩn mẩn xét lại class TestSocketServer, tìm xem có gì loại bỏ được không.

"Mày xong chưa?" Jerry hỏi.

Tôi gật đầu.

"Tốt," gã đáp lại. "Mày sắp xịt khói lỗ tai đó."

"Tôi có một câu hỏi. Chúng ta chẳng thay đổi tí nào cái SocketService. Mình thêm testSendMessage và testRecieveMessage và cả hai đều chạy. Mình lại tốn rất nhiều thời gian để viết mấy cái test và lo chuyện refactoring. Làm như thế có lợi gì cho mình nhỉ? Chúng ta chẳng thay đổi mã nguồn chính của sản phẩm gì hết!"

Jerry nhướn mày. "Bộ mày nghĩ là getBufferedReader và getPrintStream đáng được đưa vào sản phẩm?" Mấy cái này khá tầm thường; chúng chỉ hỗ trợ cho mấy cái test mà thôi.

Jerry thở dài. "Nếu mày dính vào project này, tao chỉ cho mày mấy cái test, chúng dạy mày được những gì?"

Tôi học được gì từ mấy cái test đó? Tôi học được cách tạo SocketService và gắn SocketServer từ đó. Tôi cũng học được cách gởi và nhận thông điệp. Tôi học được tên và vị trí của các classes trong framework và cách xử dụng chúng. "Ý ông mình viết mấy cái tests này để làm ví dụ cho những người khác?"

"Đó là một phần lý do, Alphonse. Những người khác sẽ có thể đọc mấy cái test này và xem cách làm việc của mã nguồn. Họ cũng có thể làm việc xuyên qua lý giải. Hơn thế, họ sẽ có thể biên dịch và thao tác các cái tests này và chứng minh với chính họ rằng cách lý giải của chúng ta đáng thuyết phục. Còn nhiều điều hơn thế nữa," gã nói tiếp, " nhưng chúng ta để dành vấn đề này cho một dịp khác."

Cùi chỏ tôi vẫn còn đau nhức nên tôi mừng là thang máy đã chữa. Trong khi đi thang máy, tôi không ngừng nghĩ ngợi: "Tests là một dạng tài liệu-có thể biên dịch được, có thể thao tác và luôn luôn đồng bộ."

[*] Còn có tên gọi là Coriolis effect gọi theo tên của kỹ sư - nhà toán học Pháp Gustave-Gaspard Coriolis. Ở đây dường như tác giả mô tả hành động tay học việc ôm cột tụt xuống và trong khi tụt xuống, anh ta ở trạng thái xoay vòng trên cột nên bị "Coriolis force". Xem thêm chi tiết ở: http://zebu.uoregon.edu/~js/glossary...fect.html. và http://satftp.soest.hawaii.edu/ocn620/coriolis/ - chú thích của người dịch.

The Crafsman 9. Dangerous Threads

Câu chuyện tay học việc trẻ tuổi của chúng ta học được bài nằm lòng: Không để các threads đeo lủng lẳng - phải nắm chắc bạn kiểm soát bước kết thúc cũng như điểm khởi tạo của chúng. Phần 9.

Robert C. Martin

Sáng nay chiếc PDA khe khẽ đánh thức tôi dậy. Cố trút cơn ngái ngủ từ não bộ, tôi tắt máy báo thức và mò vào phòng tắm. Trong khi vòi phun kỳ cọ và xoa bóp thân thể, tâm trí tôi vẩn vơ đi vào những biến cố ngày hôm trước.

Tôi trở phòng làm việc lại sau buổi giải lao, trong đầu vẫn nghĩ ngợi về giá trị thực sự từ các cú thử nghiệm. Jerry đang đợi tôi, gã nói: "Tao mừng là mày trở lại. Tao đang hoàn tất cái "test case" kế tiếp đây. Xem qua cái đi và thử đoán mục đích của nó là gì."

```
public void testMultiThreaded() throws Exception {

  ss.serve(999, new EchoServer());

  Socket s1 = new Socket("localhost", 999);

  BufferedReader br = SocketService.getBufferedReader(s1);

  PrintStream ps = SocketService.getPrintStream(s1);



  Socket s2 = new Socket("localhost", 999);

  BufferedReader br2 = SocketService.getBufferedReader(s2);

  PrintStream ps2 = SocketService.getPrintStream(s2);



  ps2.println("MyMessage");

  String answer2 = br2.readLine();

  s2.close();



  ps.println("MyMessage");

  String answer = br.readLine();

  s1.close();
```

```
  assertEquals("MyMessage", answer2);

  assertEquals("MyMessage", answer);

}
```

"Nó hơi phức tạp một chút nhưng hình như ông muốn chứng minh là SocketService có thể đối phó với hai mạch nối cùng một lúc."

"Đúng vậy," Jerry trả lời. "Mày có nhận ra là mạch nối thứ nhất lại đóng sau cùng không?"

"Không, nhưng ông nói tôi mới thấy đó. Ông làm thế để làm chi vậy?"

"Tao muốn hai phiên truy cập cùng mở liên tục," Jerry đáp.

"Tại sao?" Tôi bối rối hỏi lại. "Bởi vì khi ấy method serve trong class SocketService sẽ phải đi vào hai lần trong hai threads khác nhau, trước khi cả hai có cơ hội kết thúc," Jerry tiếp tục. "Khi một hàm được gọi vào hơn một lần trước khi nó kết thúc, cái này gọi là reentrant."

"Nhưng sao ông lại muốn test nó làm gì?" tôi cứ khăng khăng hỏi tiếp.

"Bởi vì các hàm reentrant thường đem lại cho mình những sự cố rất lý thú," Jerry mỉm cười. Tôi không hiểu nổi điều này nhưng tôi biết chắc rốt cuộc Jerry sẽ giải thích vấn đề. "OK" gã nói. "Hãy chạy thử cái test đi."

Tôi biên dịch và chạy cái test. Thanh chỉ định màu xanh lá chuyển động lẹ làng xuyên qua khung test cho chúng tôi biết rằng trọn bộ những test trước đây vẫn làm việc ngon lành. Thế rồi, trước khi kết thúc, chương trình bị khựng lại. Tôi đợi vài giây xem thử nó có thức dậy và hoàn tất hay không nhưng nó hoàn toàn treo luôn.

Sau khi nghiên cứu mã nguồn ở đoạn SocketService.serve chừng một phút, tôi nói, "Hãy xem đoạn lặp này."

```
while (running) {
  try {
    Socket s = serverSocket.accept();
    itsServer.serve(s);
    s.close();
  }
  catch (IOException e) {
  }
}
```

"itsServer.serve không trở lại để bắt lấy mạch nối thứ nhì," tôi tiếp tục. "Mạch nối thứ nhất bị treo trong đoạn EchoServer đợi mình gởi đến một thông điệp. Bởi thế chúng ta không bao giờ đi hết vòng lặp để gọi accept cho mạch nối socket thứ nhì."

Jerry cười rạng rỡ. "Khá lắm! bây giờ mình làm sao với nó đây?" "Chúng ta cần đưa itsServer.serve trong thread riêng của nó để cái vòng lặp đó có cơ hội trở lại mà không phải đợi nó."

"Lại đúng lần nữa!" gã mỉm cười. "Dám chọc nó một phát không?"

Tôi vớ lấy bàn phím và đổi method SocketService.serve như sau:

```
while (running) {
  try {
    Socket s = serverSocket.accept();
```

```
    new Thread(new ServiceRunnable(s)).start();
  }
  catch (IOException e) {
  }
}
```

Kế tiếp tôi thêm một inner class mới bên trong SocketService gọi là ServiceRunnable:

```
class ServiceRunnable implements Runnable {
  private Socket itsSocket;
  ServiceRunnable(Socket s) {
    itsSocket = s;
  }
  public void run() {
    try {
      itsServer.serve(itsSocket);
      itsSocket.close();
    }
    catch (IOException e) {
    }
  }
}
```

"Vậy là đủ rồi," tôi nói. Tôi nhấn nút test và được đền bù bằng kết quả mỹ mãn. "Nhấn nút thêm vài lần xem sao," Jerry đề nghị. "Ôi thôi, đừng chơi mấy trò này," tôi

cự nự, nhớ đến cái chứng khập khiễng lần đầu tiên chúng tôi khởi sự. Tôi miễn cưỡng chạy phần test thêm vài lần. Hiển nhiên tôi thấy ngay chỗ hỏng:

```
1) testMultiThreaded(TestSocketServer)

java.lang.NullPointerException

  at SocketService.close(SocketService.java:32)

  at TestSocketServer.

(TestSocketServer.java:30)
```

"Quỷ tha ma bắt gì đây?" tôi nhăn nhó nhìn dòng 32 của SocketService.java

```
30  public void close() throws Exception {

31    running = false;

32    serverSocket.close();

33  }
```

"Hẵng một phút," tôi chống chế. "Làm sao có thể bị null pointer exception chỗ đó được cơ chứ?" Tôi kéo lên phần TestSocketServer ở dòng 30:

```
29  public void tearDown() throws Exception {

30    ss.close();

31  }
```

"Vô lý. TearDown đóng SockerService như giả định nhưng cái serverSocket lại null là thế nào? Nếu serverSocket là null thì mình đã dính ngay lỗi từ đoạn testMultiThreaded chớ không phải trong đoạn tearDown."

Jerry hẳn cảm thấy hữu lý bởi gã nói, "Ừa."

"Jerry, cái quỷ gì đây? chẳng nghĩa lý gì cả," tôi cằn nhằn. "Cái biến serverSocket không thể là null được."

"Alphonse," Jerry nói nhỏ nhẹ. "Hãy suy nghĩ phút chốc. Trạng thái các threads thế nào?" "Hở?" tôi không bắt kịp gã. "Các cái threads," gã lặp lại một cách kiên nhẫn. "Các threads này làm gì khi tearDown được gọi?"

Tôi suy nghĩ vấn đề này chừng một phút. Rõ ràng phần test case đạt; không thì tearDown đã không được gọi. Điều này có nghĩa là cả hai mạch nối socket được tiếp nhận và serverThread đã đi xuyên vòng lặp hai lần. serverThread có thể chặn cú gọi tiếp nhận lần thứ ba hoặc giả nó chưa trở lại hàm khởi động dùng để kích thread ServiceRunnable thứ nhì.

Thread đầu của ServiceRunnable đã vào EchoServer cái này đã được đọc và viết thông điệp nhưng nó có thể chưa bị kết liễu. Nó có thể đợi phần println gởi thông điệp ngược lại từ phần test case, nhưng thread thứ nhì của ServiceRunnable hẳn có đó thời gian để kết thúc: nó đã nhận và gởi thông điệp của nó đã lâu.

Tôi giảng giải tất cả mọi điều với Jerry và gã lặng lẽ gật đầu. "Vâng," gã nói. "Tao cũng phân tích như thế." "Vậy thì sao lại có null pointer exception?" tôi hỏi, vẫn còn chút căng thẳng. "Tao chả biết," gã rụt vai. "Nhưng sự thật là nó bị đổ vỡ khi mình đóng cái serverSocket làm tao nghĩ là mình để cho một vài thread nào đó chạy làm ảnh hưởng đến thư viện socket." "Ý ông là có bug trong bộ thư viện socket?" tôi ré lên. Jerry chỉ dán mắt vào màn hình và nói, "tao không chắc; có lẽ mình dùng không đúng. Hãy đi qua một vài thử nghiệm. Điều gì xảy ra nếu cái serverThread từ phần test trước chưa đóng ngay khi chúng ta thực thi testMultiThreaded? Và rồi, khi cái close() của serverThread trước cuối cùng cũng thực thi, cái này ảnh hưởng thế nào đó đến phân đoạn close của testMultiThreaded. Mình thử nghiệm giả thuyết này sao đây?"

Tôi phải áp đặt những khái niệm này từng cái một trong não - như thường lệ, Jerry đi trước tôi nhiều bước. Nhưng một lúc sau, tôi gật đầu và đề nghị, "chúng ta có thể đợi ở phần cuối của tearDown để nắm chắc là serviceThread đóng hoàn toàn." Jerry nghĩ ngợi một giây. Với vẻ mặt rạng rỡ gã nói, "ý kiến hay đó! nếu giả thuyết của mình đúng, phần thay đổi này hẳn phải làm cho các cái test đạt mọi lần."

Tôi thay đổi như sau và chạy cái test vài chục lần. Hoàn toàn không bị hỏng nữa.

```
public void tearDown() throws Exception {
  ss.close();
  Thread.sleep(200);
}
```

Jerry mỉm cười và nói, "OK, đó là một cái test để thoả mãn giả thuyết của chúng ta. Trở ngại này dường như là một thứ ảnh hưởng nào đó giữa mấy cái test và không bị lỗi một cách cụ thể với testMultiThreaded, dẫu nó không giải thích lý do tại sao chúng ta không thấy lỗi này trước đây. Nhất định có vấn đề gì đó với testMultiThreaded làm lộ ra trạng thái này."

Tôi hơi oải với cái thay đổi cuối: "Mình không thể để cái sleep trong đó, đúng không? Đó không phải là giải pháp phải không?" tôi hỏi. "Không, nhất định không - nó chỉ là một thứ thử nghiệm mà thôi. Đem nó ra đi," Jerry trả lời.

Tôi bỏ nó ra và kiểm nghiệm không có lỗi. Thế rồi điều gì đó nảy ra trong đầu tôi. "Jerry, giả thuyết của mình không thể đúng được. Server sockets phải có khả năng đồng thời mở và đóng trong hệ điều hành, phải không? mấy cái test của mình không làm gì bất thường. Ý tôi là, thư viện socket chắc phải bị vỡ nặng nề nếu nó gián đoạn quy trình đóng mở thỉnh thoảng bị chồng lên nhau."

"Thư viện này được dùng đã lâu. Tao không nghĩ là nó bị vỡ đâu," Jerry ngấm ngoẳng. "Nhất định phải có gì đặc biệt trong cách chúng ta viết mấy cái test làm cho thư viện phản ứng như thế này." "Có thể nào do chúng ta dùng cùng một cổng số?" tôi hỏi.

Tôi đổi trọn bộ các test dùng cổng số khác nhau. Sau khi qua hàng chục test, tôi nói, "nó sẽ không hỏng. Vấn đề nằm ở chỗ nhiều tests cùng dùng một cổng số." "Đây là điều rất lý thú," Jerry trả lời. "Điều đó giải thích lý do tại sao mình không thấy lỗi này trên những hệ thống khác. Các hệ thống không dùng cùng một cổng số."

Tôi lại thấy oải nữa. "Jerry, đây cũng chưa phải là giải pháp tốt cho mình. Chúng ta phải tìm cách làm sao cho SocketService để ngăn ngừa trở ngại này, phải không?" "Tuyệt đối là như vậy rồi Alphonse. Vậy thì tiến hành đi và để cổng số y như cũ và tính thử mình phải làm gì."

Ngay khi test có lỗi trở lại, tôi nhìn Jerry, đợi chờ. "OK, mình xử cái quỷ này sao đây?" "Chúng ta không cho phép SocketService.close trở về cho đến khi serverThread kết thúc," gã nói, vớ lấy bàn phím và thay đổi như sau:

```
public void close() throws Exception {
  if (running) {
    running = false;
    serverSocket.close();
    serverThread.join();
  }
  else {
    serverSocket.close();
  }
}
```

Sau hàng tá test, gã nói, "Ừa, đâu vào đấy." "Tôi đoán bài học ở đây là: đừng để threads treo lủng lẳng. Phải nắm chắc mình kiểm soát được bước kết thúc cũng như điểm khởi tạo của chúng," Tôi nói. "Đó là một bài học nằm lòng rất tốt," gã trả lời. "Một thread lủng lẳng có thể gây tai hoạ khi mày ít ngờ đến nhất."

The Crafsman 10. Intergation Unbound

Những vòng xoay vô giới hạn

Nếu có một vòng xoay động, bạn không muốn đổi tập họp hiện có, nhất là từ một thread khác. Phải chăng "design patterns" là giải pháp cho bạn?

Robert C. Martin

Mỗi tháng tôi lại dùng điểm tâm một lần ở đài quan sát. Đây là điều ngoại hạng cho tay học việc như tôi, tôi khoái ăn dưới vòm trời mở rộng. Trong lúc ăn, tôi ngẫm nghĩ về chuyện thread treo lủng lẳng được chúng tôi giải quyết ngày hôm qua. Chúng tôi sửa cái serverThread nhưng lại để trọn bộ các thread thuộc serviceRunnable treo tòng teng. Tôi biết thế nào Jerry cũng muốn sửa mấy cái ấy cho sớm.

Đúng y như vậy, ngay khi tôi bước vào phòng làm việc, Jerry đã mang mấy cái test case trên màn hình như sau:

```
public void testAllServersClosed() throws Exception {
  ss.serve(999, new WaitThenClose());
  Socket s1 = new Socket("localhost", 999);
  Thread.sleep(20);
  assertEquals(1,WaitThenClose.threadsActive);
  ss.close();
  assertEquals(0, WaitThenClose.threadsActive);
}
```

"Ông phải chắc ăn trọn bộ những cái SocketServers đóng hết ngay khi trở lại từ bước đóng SockeService," tôi nói.

"Tao muốn chắc ăn là mình không để cho mấy cái servers đó treo lủng lẳng như thế," Jerry trả lời.

"Nhưng ông chỉ test nó với một server thôi mà," tôi đáp lại. "Bộ mình không cần test với nhiều server hay sao?"

"Đúng thế!" Jerry mỉm cười. "Nhưng hãy làm xong cái test này ngon lành cái đã."

"OK," tôi trả lời. "Tôi biết cách viết WaitThenClose ra sao rồi."

```
class WaitThenClose implements SocketServer {
 public static int threadsActive = 0;

 public void serve(Socket s) {
  threadsActive++;
  delay();
  threadsActive--;
 }
 private void delay() {
  try {
   Thread.sleep(100);
  }
  catch (Interrupted Exception e) {
  }
 }
}
```

Jerry gật gù trong lúc mã nguồn của tôi hiện ra trên màn hình; cái WaitThenClose của tôi đúng y như gã dự tưởng. Tôi biên dịch mã nguồn này và chạy mấy phần test, chúng hỏng như dự đoán:

```
1)  testAllServersClosed AssertionFailedError:

    expected:<0> but was:<1>
```

Jerry xoa tay và nói, "bây giờ hãy làm cho nó đạt đi." Gã với lấy bàn phím nhưng tôi cản gã lại. "Tôi nghĩ là tôi có một ý kiến". Thế nên, trong khi Jerry quan sát, tôi thay đổi đoạn mã như sau:

```
private LinkedList serverThreads = new LinkedList();


public void serve(int port, SocketServer server) throws Exception {
 itsServer = server;
 serverSocket = new ServerSocket(port);
 serverThread = new Thread(
  new Runnable() {
   public void run() {
    running = true;
    while (running) {
     try {
      Socket s = serverSocket.accept();
      Thread serverThread = new Thread(new ServiceRunnable(s));
      serverThreads.add(serverThread);
      serverThread.start();
```

```
        }
        catch (IOException e) {
        }
       }
      }
    }
  );
  serverThread.start();
}

public void close() throws Exception {
  if (running) {
    running = false;
    serverSocket.close();
    serverThread.join();
    for (Iterator i = serverThreads.iterator(); i.hasNext();) {
      Thread thread = (Thread) i.next();
      serverThreads.remove(thread);
      thread.join();
    }
  }
  else {
    serverSocket.close();
  }
}
```

Khi mã nguồn được biên dịch, Jerry nhăn nhó. "Vậy được không?" tôi hỏi.

"Hãy xem nào," gã trả lời. "Chạy thử cái test xem sao."

Khi chạy cái test, bị một lỗi khác thường:

1) testOneConnection java.util.ConcurrentModificationException

"Cái gì vậy?" tôi hỏi.

"Mày làm vỡ luật đó, Alphonse," Jerry nói. "Không bao giờ thêm hoặc bớt từ một cái list trong khi mày có một vòng xoay động."

"Tất nhiên rồi!" tôi nói một cách ngượng ngùng. "Ok, nhưng chuyện này dễ sửa thôi, bởi vì tôi không cần phải tháo bỏ cái cái thread từ list." Tôi bỏ dòng remove và chạy đoạn test lại. "À! bây giờ thì nó chạy."

Jerry gật đầu nhưng nhìn tôi chằm chặp một cách chờ đợi. "Gì hở?" tôi gào lên sau nửa phút chịu đựng kiểu nhìn của gã. "Mày vẫn đang thay đổi cái list trong khi vòng xoay ứng động," gã phán.

"Vậy sao?" tôi quả thật bối rối. "Chỉ có một nơi duy nhất cái list được thay đổi, và đó là nơi thread được thêm vào trong running loop. Làm sao nó được gọi trong khi vòng xoay động?"

"Có thể được," Jerry nói. "Cú gọi để tiếp nhận có thể ở tình trạng chực trở lại ngay khi mày đi vào vòng xoay. Khi vòng xoay chặn một cú nối (join), phần tiếp nhận sẽ trở lại và thêm một thread nữa vào list."

"OK, nhưng mình test chuyện đó được không?" tôi hỏi.

"Mình có thể làm được chuyện đó nhưng chẳng ích gì," Jerry trả lời. "Hoá ra ở một nơi khác nơi mày sẽ thay đổi cái list trong khi vòng xoay mở ra."

"Có à?"

"Ừa, mày sắp sửa thêm nó vào đó," Jerry mỉm cười. Gã nói tiếp, "Có bao nhiêu thread trong list đó vậy?"

"Cả lũ... eo ôi!" tôi vỗ trán. "Tôi nên bỏ cái thread ra khỏi list khi nó đã hoàn thành công tác! không thì, các thread đã hoàn tất sẽ đeo tòng teng trong list." Tôi với lấy bàn phím và thay đổi như sau:

```
class ServiceRunnable implements Runnable {
 private Socket itsSocket;

 ServiceRunnable(Socket s) {
  itsSocket = s;
 }

 public void run() {
  try {
   itsServer.serve(itsSocket);
   serverThreads.remove(Thread.currentThread());
   itsSocket.close();
  }
```

```
    catch (IOException e) {
    }
  }
}
```

"À há, bây giờ nó lại hỏng tiếp," tôi nói. "Ông nói đúng lắm - vài cái thread hoàn tất trước khi vòng xoay chấm dứt. Cha chả, vòng xoay "ý kiến" với các cập nhật liên đới quả là điều thật hay!"

"Đúng thế," Jerry gật đầu. "Bây giờ để tao chỉ mà cách tao trị nó như thế nào."

```
public void close() throws Exception {
  if (running) {
    running = false;
    serverSocket.close();
    serverThread.join();
    while (serverThreads.size() > 0) {
      Thread t = (Thread)serverThreads.get(0);
      serverThreads.remove(t);
      t.join();
    }
  }
  else {
    serverSocket.close();
  }
```

}

"Rồi!" Jerry nói. "Bây giờ thì mấy cái test hẳn phải đạt."

"Tôi biết rồi," tôi thốt ra. "Thay vì dùng vòng xoay, ông chỉ kéo phần tử thứ nhất ra khỏi list và tiếp tục lặp lại cho đến khi list trống rỗng."

"Đúng đó," Jerry trả lời. "Bằng cách đó, không vòng xoay nào mở ra quá lâu. Các cú nối (joins) có thể mất thời gian, cho nên để vòng xoay mở quá lâu khi các thread khác thay đổi list là điều không hay."

"Thế, mình xong việc rồi sao?" Jerry lắc đầu. "Không, vẫn còn hiểm nguy," gã cảnh báo.

"Ý ông thế nào vậy?" tôi ré lên, thất vọng. "Chớ có sự cố gì nữa đây?"

"Alphonse, mỗi khi mày có một container bị nhiều thread thay đổi, rất có cơ hội hai thread va nhau bên trong container. Một thread có thể thêm một phần tử trong khi một thread khác lại xoá phần tử khác. Khi trường hợp này xảy ra, container có thể bị hỏng và những chuyện kỳ quái có thể xảy ra."

"Vậy ý ông là mình nên đồng bộ hoá truy cập đến container?" tôi hỏi.

"Chính xác," Jerry trả lời. "Chúng ta cần nắm chắc không có thread nào khác có thể truy cập container trong khi nó bị thay đổi."

"Đơn giản thôi," tôi nói trong khi gom lại đoạn thêm và hai đoạn bớt với biện thức đồng bộ (serverThreads) {...}. Tôi chạy mấy cái tests và chúng đạt hết.

"Đó là một cách," Jerry nói với nụ cười trên mặt, "nhưng nó hơi bị dễ dính lỗi. Nếu có ai chỉnh sửa mã nguồn và đặt vào một cái add hay remove, họ phải nhớ đặt phần đồng bộ hoá vào. Nếu họ quên, những chuyện tồi tệ có thể xảy ra."

Tôi ngẫm nghĩ vấn đề ấy vài phút và xác định gã nói đúng - nếu chúng ta không cần phải gom các dòng thao túng list bằng biện thức đồng bộ thì có lẽ tốt hơn. "Thế cách nào tốt hơn vậy?"

"Tao chỉ cho mày xem." Gã lấy bàn phím và tháo bỏ các dòng synchronized của tôi. Sau đó gã thay đổi thêm một dòng mã nữa - dòng tạo LinkedList ngay lúc đầu:

```
private List serverThreads = Collections.synchronizedList( new LinkedList());
```

Jerry biên dịch mã nguồn và chạy trọn bộ các cái test. Mọi sự ổn cả. Sau rồi gã hỏi, "Mày biết gì về design patterns hả Alphonse? Có bao giờ mày nghe đến Decorator pattern chưa?"

"Tất nhiên là tôi nghe về chúng rồi, và tôi cũng thấy sách nói về chuyện này trên giá sách của thiên hạ, nhưng tôi không biết nhiều lắm về chúng."

Jerry nhìn tôi nghiêm khắc nói, "vậy thì đến lúc mà nên bắt đầu học về chúng một cách nghiêm chỉnh đi. Mày có thể mượn sách của tao và nghiên cứu nó nếu thích. Đầu tiên tao muốn mày đọc chương nói về Decorator pattern. Hàm synchronizedList mình vừa gọi để gói cái LinkedList trong một Decorator. Mọi cú gọi đến LinkedList đều được nó đồng bộ hoá cả."

"Nghe đúng là một giải pháp hay," tôi đáp. "Ừa, mà mày cũng phải nhớ đồng bộ hoá cụ thể những nơi dùng vòng xoay." Jerry cau mày.

"Vậy sao?" Tôi hỏi. "Ý ông vòng xoay không được đồng bộ hoá trong danh sách đồng bộ sao?"

"TANSTAAFL," gã trả lời.

"Hở?" tôi hỏi, thộn người ra. Không biết có phải gã nói tiếng Clangrish hay gì đây.

"TANSTAAFL," gã lặp lại theo kiểu khống chế; rồi gã mỉm cười. "There Ain't No Such Thing As A Free Lunch" (Không hề có cái gọi là buổi ăn trưa miễn phí).

"Tôi biết," tôi mỉm cười trong khi rảo bước về buồng của tôi.

(Còn nữa.....)

The Crafsman 11. Forget The main()

Quên đi hàm main()

Chúng tôi đã dựng xong chương trình để gọi phần biên dịch SMC từ xa, gởi mã nguồn đến server và gởi ngược lại hồ sơ đã biên dịch. Thế nhưng tại sao Jerry lại khăng khăng test mã nguồn lẻ tẻ?

Robert C. Martin

Trong đầu tôi cứ cân nhắc mãi mớ threads treo tòng teng trong khi ăn món mì ống spaghetty một cách lơ đãng. Sau bữa trưa, tôi trở về phòng làm việc tìm Jerry.

"Ông C nghĩ là SocketServer sẵn sàng để dùng rồi đó, và bây giờ ông ta muốn chúng mình làm việc với ứng dụng SMSRemote."

"Ồ, đúng nhỉ!" Tôi nói. "Thì đó là lý do có SocketServer mà - mình đã dựng xong chương trình dùng để gọi phần biên dịch SMC từ xa, gởi mã nguồn đến server và gởi ngược lại hồ sơ đã biên dịch."

Jerry nhìn tôi chờ đợi và hỏi, "mày nghĩ mình khởi công sao đây?"

"Tôi nghĩ là tôi cần biết người dùng sẽ sử dụng chúng ra sao cái đã," tôi trả lời.

"Xuất sắc!" gã mỉm cười. "Khởi đầu từ cái nhìn của người dùng luôn luôn là một điều hay. Thế thì cách nào là cách đơn giản nhất người dùng có thể mó đến tiện ích này?"

"Anh ta có thể yêu cầu một hồ sơ nào đó được biên dịch," tôi trả lời. "Lệnh ấy có thể như thế này." tôi viết lên tường như sau: java SMCRemoteClient myFile.sm

"Coi được đó," Jerry nói. "Mình bắt đầu sao đây?"

Tôi cảm thấy khá vững tin sau khi làm SocketServer chạy được, thế nên tôi vớ lấy bàn phím và bắt đầu gõ:

```
public class SMCRemoteClient {

  public static void main(String args[]) {

    String fileName = args[0]; }

  }

}
```

"Mày có cái test cho nó không?" Jerry ngắt ngang.

"Ý ông là sao?" tôi hỏi một cách thiếu kiên nhẫn. "Mã nguồn này thuộc dạng lẻ tẻ - sao mình phải viết cái test cho nó làm chi?"

"Nếu mày không viết một cái test cho nó thì làm sao mày biết là có cần hay không?" gã hỏi.

Câu hỏi ấy làm tôi khựng lại. "Tôi nghĩ điều ấy quá hiển nhiên," sau rốt tôi nói.

"Vậy sao?" Jerry trả lời. "Tao không được thuyết phục cho lắm. Hãy thử một lối khác xem sao." Gã vớ lấy bàn phím và xoá hết mã nguồn của tôi. Tự ái trong lòng bùng lên nhưng tôi cố dằn nó xuống. Dù gì cũng chỉ có vỏn vẹn bốn dòng code mà thôi.

"OK, mình cần những hàm nào đây?" gã hỏi. Tôi nghĩ ngợi vài giây và nói, "mình cần lấy tên hồ sơ từ dòng lệnh nhưng tôi không biết ông sẽ làm sao nếu không có phần mã nguồn ông vừa xoá mất."

Jerry nhìn tôi với vẻ chế giễu, hắn nói, "tao biết," và bắt đầu gõ phím. Đầu tiên gã viết một đoạn test framework quen thuộc:

```
import junit.framework.*;

public class TestSMCRemoteClient extends TestCase {

  public TestSMCRemoteClient(String name) {

    super(name);

  }

}
```

Gã biên dịch và chạy thử, nắm chắc phần test phải hỏng vì thiếu tests, và rồi gã thêm đoạn test sau:

```
public void testParseCommandLine() throws Exception {
  SMCRemoteClient c = new SMCRemoteClient();
  c.parseCommandLine(new String[]{“filename”});
  assertEquals(“filename”, c.filename());
}
```

"OK," tôi nói. "Có vẻ như ông lấy đối số của dòng lệnh bằng function parseCommandLine thay vì dùng main, nh ưng phiền như thế làm gì?"

"Thế để tao có thể thử nghiệm," Jerry cố nín cười, trả lời.

"Nhưng chẳng có gì để mà thử cả," tôi cằn nhằn.

"Điều đó có nghĩa quá hời để viết phần test," gã cười toe toét.

Tôi biết tôi sẽ không thắng nổi trận đấu này nên đành thở dài, vớ lấy bàn phím và viết đoạn mã sau để phần test có thể đạt:

```
public class SMCRemoteClient {
  private String itsFilename;

  public void parseCommandLine(String[] args) {
    itsFilename = args[0];
  }

  public String filename() {
```

```
    return itsFilename;
  }
}
```

Jerry gật đầu và lặng lẽ viết phần test case kế tiếp.

```
public void testParseInvalidCommandLine() {
  SMCRemoteClient c = new SMCRemoteClient();
  boolean result = c.parseCommandLine(new String[0]);
  assertTrue("result should be false", !result);
}
```

Lẽ ra tôi phải biết gã chỉ cho tôi lý do tại sao tôi nghĩ, viết một cái test không cần thiết lại là một khái niệm hay. "OK", tôi thú nhận. "Tôi đoán việc lấy đối số của dòng lệnh ít vụn vặt hơn là tôi nghĩ. Có lẽ nó đáng để có một cái test cho riêng nó." Thế rồi tôi vớ lấy bàn phím và làm cho phần test đạt.

```
public boolean parseCommandLine(String[] args) {
  try {
    itsFilename = args[0];
  }
  catch (ArrayIndexOutOfBoundsException e) {
    return false;
  }
  return true;
}
```

Cân nhắc kỹ lưỡng, tôi refactor biến số c và khởi động nó trong function setUp. Các tests đều đạt. Trước khi Jerry có thể đề nghị phần test case tiếp theo, tôi nói, "Rất có khả năng hồ sơ không tồn tại. Chúng ta nên viết một cái test chứng minh mình có thể lo cho trường hợp ấy được."

"Quả vậy," Jerry nói trong khi tóm lấy bàn phím trong tay tôi. "Nhưng để tao chỉ cho mà cách tao khoái làm thế nào."

```
public void testFileDoesNotExist() throws Exception {
  c.setFilename("thisFileDoesNotExist");
  boolean prepared = c.prepareFile();
  assertEquals(false, prepared);
}
```

"Mày thấy không?" gã giảng giải. "tao muốn ước định mỗi đối số của dòng lệnh trong function của chính nó thay vì nhập chung cả mớ mã phân tích và ước định chung với nhau." Trong khi đó, tôi kín đáo đảo mắt ráng ghi nhớ những điếm ấy để tham khảo sau này, tôi lấy bàn phím và thay đổi những điểm sau để làm cho phần test đạt:

```
public void setFilename(String itsFilename) {
  this.itsFilename = itsFilename;
}
```

```
public boolean prepareFile() {
  File f = new File(itsFilename);
  if (f.exists())
    return true;
  else
    return false;
```

```
}
```

Trọn bộ các test đều đạt. Jerry nhìn tôi rồi nghía sang bàn phím. Hiển nhiên gã muốn "lái" bàn phím. Hôm nay dường như gã tràn đầy sáng kiến, bởi thế tôi chuyển bàn phím về phía gã.

"OK, bây giờ xem đây!" gã nói, cỗ máy trong gã rõ ràng đang gầm rú.

```
public void testCountBytesInFile() throws Exception {
  File f = new File(“testFile”);
  FileOutputStream stream = new FileOutputStream(f);
  stream.write(“some text”.getBytes());
  stream.close();

  c.setFilename(“testFile”);
  boolean prepared = c.prepareFile();
  f.delete();
  assertTrue(prepared);
  assertEquals(9, c.getFileLength());
}
```

Sau khi nghiên cứu mã nguồn của gã vài giây, tôi trả lời, "Ông muốn preparFile() để lấy độ dài của hồ sơ? tại sao?"

"Tao nghĩ lát nữa mình sẽ cần chúng," gã giải thích. "và đó là một cách hay để chứng minh mình có thể đối phó với một hồ sơ hiện có."

"Mình cần nó để làm gì kia chớ?" tôi nằn nặc.

"Chúng ta sẽ phải gởi nội dung của hồ sơ xuyên qua socket đến server, phải không?" Jerry hỏi.

"Vâng."

"Và chúng ta cần biết sẽ gởi bao nhiêu chữ," gã kiên nhẫn giải thích.

"Hườm... có lẽ," tôi miễn cưỡng trả lời.

"Tin tao đi," gã mỉm cười. "tận cùng thì tao làm người hướng đạo cơ mà."

"OK, khỏi nói đến chuyện ấy," tôi trả lời một cách thiếu kiên nhẫn. "Tạo sao ông lại tạo hồ sơ trong phần test kia chớ? sao ông không giữ hồ sơ này sẵn thay vì lần nào cũng phải tạo nó ra?"

Jerry cười khẩy rồi trở nên nghiêm túc. "Tao ghét giữ lại các nguồn bên ngoài cho mấy cái test. Bất cứ khi nào có thể được, tao để cho mấy cái test tạo ra nguồn chúng cần. Với cách ấy, không cách nào tao bị mất nguồn cả, hoặc ngay cả trường hợp nguồn bị hỏng nữa."

"Ồ, điều này thì quả có lý," tôi thừa nhận, "nhưng tôi vẫn không điên khùng với mấy thứ độ dài của hồ sơ kia."

"Nhớ đó. Mày sẽ thấy!"

Tôi lấy bàn phím và bắt đầu làm việc với phần cho phép đoạn test đạt. Trong khi tôi gõ phím, tôi thấy hơi lạ vì tôi đang viết mã chính trong khi thiết kế là của Jerry -

nhưng những gì Jerry làm chỉ là viết những đoạn test case nhỏ. Bạn có thể thực sự xếp loại một thiết kế bằng cách viết những test case hay không?

```
public long getFileLength() {
  return itsFileLength;
}

public boolean prepareFile() {
  File f = new File(itsFilename);
  if (f.exists()) {
    itsFileLength = f.length();
    return true;
  }
  else
    return false;
}
```

Test case kế tiếp tạo ra một cái server giả và dùng để thử khả năng của SMCRemoteClient truy cập vào đó.

```
public void testConnectToSMCRemoteServer() throws Exception {
  SocketServer server = new SocketServer(){
```

```
    public void serve(Socket socket) {
      try {
      socket.close();
      }
      catch (IOException e) {
      }
    }
  };
  SocketService smc = new SocketService(SMCPORT, server);
  boolean connection = c.connect();
  assertTrue(connection);
}
```

Với rất ít trở ngại, tôi làm cho phần test case đạt:

```
public boolean connect() {
  try {
    Socket s = new Socket("localhost", 9000);
    return true;
  }
  catch (IOException e) {
  }
  return false;
}
```

"Tuyệt!" Jerry nói. "Mình nghĩ giải lao một tí."

"OK," tôi trả lời, "nhưng hãy viết phần main() trước đã."

"main() gì, dính dự gì ở đây?" gã hỏi.

"Hở? đó là main của chương trình chớ gì!"

"Thế thì sao chớ?" Jerry rụt vai. "Nó chỉ gọi parseCommandLine(), parseFile() và connect(). Còn lâu lắm mình mới test mấy thứ đó!"

Tôi rời phòng làm việc và đi về phía phòng giải lao. Trước giờ tôi cứ nghĩ main() là function đầu tiên cần được viết, nhưng Jerry rất đúng. Rốt cuộc, main() chỉ là một function khá thiếu thú vị.

Còn tiếp.....

The Crafsman 12. Three Ugly Lines

Ba dòng xấu xí

Robert C. Martin

Tôi nghỉ giải lao trên đài quan sát. Khi lớp chắn bằng nước đá đi xuyên qua vùng phân tử dày cộm làm cho lớp nước đá nhập nhoè trong những làn chớp xanh và những mô hình chuyển biến khắp bề mặt của lớp chắn - làm tôi nhớ đến Bắc cực quang của trái đất.

Như thường lệ, Jerry đang đợi tôi sau buổi giải lao. Gã nhìn tôi và nói, "OK, hãy gởi một hồ sơ qua socket."

"Ông xem cuộc trưng bày Cherenkov -1- chưa?

"Tuyệt đẹp!" gã cười mỉm. Tôi đoán lâu lâu gã cũng nghỉ giải lao - nhưng thường thường gã đi đâu?

"OK," tôi nói. "Tôi sẽ viết phần test case." Phần đầu tiên tôi gõ là đoạn mã dùng để tạo hồ sơ được gởi qua socket.

```
public void testSendFile() throws Exception {
  File f = new File("testSendFile");
  FileOutputStream stream = new FileOutputStream(f);
  stream.write("I am sending this file.".getBytes());
  stream.close();
}
```

"Tôi biết ông muốn tạo hồ sơ dữ liệu trong mã test hơn là phụ thuộc vào tình trạng chúng có hiện diện hay không," tôi nói.

"Đúng thế," Jerry trả lời. "Nhưng mày có thấy mày bị mấy thứ lặp lại không?"

Tôi xem lại phần test và thấy ngay chúng tôi viết đoạn mã gần như trùng lặp với method testCountBytesInFile() mà chúng tôi đã hoàn thành trước giờ giải lao. "Chỉ có bốn dòng code mà thôi," tôi nói.

"Đúng thế," Jerry đáp. "Nhưng sự trùng lặp nên được vứt bỏ ngay khi có thể được. Không thì mày sẽ ôm một mớ code khổng lồ đầy mập mờ và đầy lỗi."

"Được rồi," tôi trả lời, "sửa cái này dễ thôi." Tôi tỉa tót một hàm mới gọi là createTestFile() và thay đổi cả testCountBytesInFile() lẫn testSendFile() để gọi hàm này.

```
private File createTestFile(String name, String content)  throws IOException {

  File f = new File(name);

  FileOutputStream stream =  new FileOutputStream(f);

  stream.write(content.getBytes());

  stream.close();

  return f;

}
```

Tôi chạy thử cái test để chắc ăn là không làm hỏng gì cả, rồi tiếp tục viết phần test. Tôi biết nó cần giả lập main(), cho nên tôi gọi những hàm main() cần gọi. Thế rồi tôi thêm vào phần gọi cuối để gởi hồ sơ đi.

```
public void testSendFile() throws Exception {

  File f = createTestFile("testSendFile", "I am sending this file.");


  c.setFilename("testSendFile");

  assertTrue(c.connect());
```

assertTrue(c.prepareFile());

assertTrue(c.sendFile());

}

"Tốt," Jerry gật đầu. "Mày liệu trước là mình sẽ cần một method phía client có tên là sendFile().

"Đúng vậy," tôi nói. "Method này sẽ gởi hồ sơ nào được chuẩn bị trước."

Tôi trở lại với phần test và bị trở ngại. Làm sao tôi kiểm nghiệm được hồ sơ tôi tạo ra và "gởi đi" thật sự được gởi đến server trong khi chúng tôi chẳng có hồ sơ nào? Phải chăng tôi cần viết cả phần server trước khi tôi có thể kiểm nghiệm chuyện này? Tôi định test gì vậy nhỉ?

Tôi bực dọc ngồi yên trong khi Jerry nhìn tôi chờ đợi. Thế rồi khi tôi xoay qua và giải thích điểm khó khăn. Gã giải thích "Không, mày không cần phải viết cái server," "Chúng ta chỉ test mỗi khả năng gởi hồ sơ của client, chớ chẳng phải khả năng nhận hồ sơ của server."

"Nhưng làm sao tôi gởi hồ sơ trong khi chẳng có server để nhận?"

"Mày có thể tạo ra "stub" server chỉ làm tối thiểu công việc mày cần thôi," Jerry trả lời. "Nó chẳng cần phải thực sự nhận hồ sơ - nó chỉ tiếp báo là mày đã gởi hồ sơ đúng cách."

"Hừm... như thế này chăng?"

assertTrue(server.fileReceived);

"Như vậy được rồi," Jerry gật đầu. "Bây giờ làm cho cái test đạt đi." Tôi nghĩ về vấn đề này và nhận ra nó không quá khó, thế nên tôi viết một cái server giả chẳng làm gì hết:

```
class TestSMCRServer implements SocketServer {

  public boolean fileReceived = false;

  public void serve(Socket socket) {

  }

}
```

Jerry nói, "À, lại thêm trùng lặp!" Tôi xem lại và thấy trước đoạn ngắt, chúng tôi đã ứng hiệu server giả tương tự trong method testConnectToSMSRemoteServer() - thế nên tôi loại bỏ nó.

Thế rồi tôi bắt đầu phần server với method SetUp() trong phần test và đóng nó bằng method TearDown(). Trước khi mỗi method của test được gọi, server khởi động; khi method của test trả ngược về, nó đóng lại.

```
protected void setUp() throws Exception {

  c = new SMCRemoteClient();


  server = new TestSMCRServer();

  smc = new SocketService(SMCPORT, server);

}


protected void tearDown() throws Exception {

  smc.close();

}
```

Cuối cùng tôi viết method giả sendFile() trong SMCRemoteClient:

```
public boolean sendFile() {
 return false;
}
```

Mấy cái test bị hỏng. Tôi thở dài. "Làm gì bây giờ?"

"Gởi cái hồ sơ đi," gã ra lệnh.

"Chỉ mở hồ sơ ra và tống nó qua socket sao?" Tôi hỏi.

"Không, có lẽ mình cần cho server biết để tiếp nhận hồ sơ - cho nên hãy gởi một thông điệp đơn giản và gởi tiếp theo đó phần hồ sơ." Jerry thay đổi SMCRemoteClient như sau.

"Đầu tiên, mình cần lấy cái "stream" ra từ socket," gã nói.

```
public boolean connect() {
 boolean connectionStatus = false;

 try {
  Socket s = new Socket("localhost", 9000);
  is = new BufferedReader(new InputStreamReader(s.getInputStream()));
  os = new PrintWriter(new OutputStreamWriter(s.getOutputStream()));
```

```
    connectionStatus = true;
  }
  catch (IOException e) {
    e.printStackTrace();
    connectionStatus = false;
  }
  return connectionStatus;
}
```

"Rồi," Jerry tiếp tục, "để đọc được hồ sơ phải có sẵn."

```
public boolean prepareFile() {
  boolean filePrepared = false;

  File f = new File(itsFilename);
  if (f.exists()) {
    try {
      itsFileLength = f.length();
      fileReader = new BufferedReader(
        new InputStreamReader(new FileInputStream(f))
      );
      filePrepared = true;
    }
    catch (FileNotFoundException e) {
      filePrepared = false;
```

```
    e.printStackTrace();
   }
  }
  return filePrepared;
}
```

"Sau cùng," gã nói, "chúng ta có thể gởi hồ sơ."

```
public boolean sendFile() {
  boolean fileSent = false;
  try {
    writeSendFileCommand();
    fileSent = true;
  }
  catch (Exception e) {
    fileSent = false;
  }
  return fileSent;
}
private void writeSendFileCommand() throws IOException {
  os.println("Sending");
  os.println(itsFilename);
  os.println(itsFileLength);
```

```
  char buffer[] = new char[(int) itsFileLength];

  fileReader.read(buffer);

  os.write(buffer);

  os.flush();

}
```

"Ôi!" tôi nói. "Gõ ra cả đống mà chẳng thử nghiệm." Jerry nhìn tôi một cách ngượng ngịu. "Ừa, tao cũng run lắm." Gã nhấn nút test và phần test bị hỏng vì server.fileRecieved trả lại sai. "Ui cha!" Jerry nói. "Mình tránh mạch đập Muon -2- đó!"

"Thế," tôi nói, bắt chước giọng thật giống Dr. Watson, "bạn sắp sửa tiến hành hồ sơ với ba dòng. Dòng thứ nhất gồm string "Sending", dòng thứ hai gồm tên hồ sơ và dòng thứ ba gồm chiều dài của hồ sơ. Sau đó, bạn gởi hồ sơ theo dạng chuỗi ký tự."

"Rồi," Jerry mỉm cười. "Tao đã nói với mày trước buổi giải lao là mình cần chiều dài của hồ sơ rồi mà."

"Hừm, tôi đoán thế, Sherlock," tôi nói một cách miễn cưỡng.

"Bây giờ mình chỉ cần nhận hồ sơ từ server giả. Mày muốn thử một phát không?" Jerry hỏi. Tôi khá chắc nên phải làm gì nên đầu tiên tôi đổi cái test để chắc ăn chúng tôi có tên hồ sơ, chiều dài và nội dung hồ sơ:

```
public void testSendFile() throws Exception {

  File f = createTestFile("testSendFile", "I am sending this fil e.");

  c.setFilename("testSendFile");

  assertTrue(c.connect());

  assertTrue(c.prepareFile());
```

```
  assertTrue(c.sendFile());

  Thread.sleep(50);
  assertTrue(server.fileReceived);
  assertEquals("testSendFile", server.filename);
  assertEquals(23, server.fileLength);
  assertEquals("I am sending this file.", new String(server.content));
  f.delete();
}
```

Kế tiếp tôi đổi cái server giả cho nó phân giải dữ liệu vào và bảo đảm thực tính:

```
class TestSMCRServer implements SocketServer {
  public String filename = "noFileName";
  public long fileLength = -1;
  public boolean fileReceived = false;
  private PrintStream os;
  private BufferedReader is;
  public char[] content;
  public String command;

  public void serve(Socket socket) {
    try {
      os = new PrintStream(socket.getOutputStream());
      is = new BufferedReader(new InputStreamReader(socket.getInputStream()))
```

```
      os.println("SMCR Test Server");
      os.flush();
      parse(is.readLine());
    }
    catch (Exception e) {
    }
  }
  private void parse(String cmd) throws Exception {
    if (cmd != null) {
      if (cmd.equals("Sending")) {
        filename = is.readLine();
        fileLength = Long.parseLong(is.readLine());
        content = new char[(int)fileLength];
        is.read(content,0,(int)fileLength);
        fileReceived = true;
      }
    }
  }
}
```

Cuối cùng tôi điều chỉnh SMCRemoteClient.connect() để nó đợi thông điệp SMCR được gởi từ server giả:

```
public boolean connect() {
  ...
  String headerLine = is.readLine();
```

```
  connectionStatus = (headerLine != null) && headerLine.startsWith("SMCR");

  ...

}
```

Tôi không gõ hết những thứ trên cùng một lúc. Tôi thay đổi từng bước nhỏ, chạy test giữa mỗi thay đổi. Tôi biết Jerry có ấn tượng tốt, đặc biệt vì gã còn bị quê chuyện thay đổi to lớn ở trên. Sau cùng, khi mọi test đều đạt, tôi cảm thấy hơi cha nội hơn một tí, tôi đánh liều bằng một nhận xét.

"Jerry," tôi nói. "Đoạn code này xấu xí quá."

"Ý mày thế nào?" gã hỏi.

"Hèm, gởi ba dòng: tên hồ sơ, chiều dài và dòng "Sending"."

Jerry nhìn tôi một cách nhún nhường. "Cứ cho là mày biết cách hay hơn."

"Tôi nghĩ thế." tôi mỉm cười và bắt đầu gõ....

-1- Cherenkov display: thuộc nghiên cứu vật lý cao cấp. Một đề tài hết sức thú vị và được nhiều nhóm nghiên cứu quan tâm. Có một pdf phân tích Chrenkov display rất cụ thể ở: www.lip.pt/~varela/projfc/Showers/Auger-3.pdf. Ngoài ra còn có vô số tài liệu về vấn đề này trên Internet cho những ai thích đào sâu.

-2- Muon: Một "muon" là một phân tố không ổn định trong vùng phản xạ gần bề mặt trái đất. Nó có trọng lượng hơn 207 lần trọng lượng một electron và tồn tại trong cả thể dạng âm hoặc dương.

The Crafsman 13. The Bester Solution

Một giải pháp tốt hơn

Trong khi làm việc với bài tập SocketServer, Alphonse khám phá ra việc chuyển tải objects đơn giản và hiệu xuất hơn chuyển tải strings - phải chăng anh ta đã "Micahed" -1- kẻ du hành của chàng?

Robert C. Martin

Ở mức .045 hiện tại, *Cái đích* vẫn là những phần đời của tương lai. Mỗi thế hệ từ lúc khởi hành cảm như những phần đời kéo dài vô tận trước khi chúng xảy ra. Đôi khi cảm giác này đầy tuyệt vọng - nhưng hôm nay không phải thế. Hôm nay, tôi dùng một ít thời gian vô tận ấy để chế diễu Jerry.

Tôi duỗi tay ra phía trước bàn phím và bẻ mấy khớp tay. Tôi lắc lư cái đầu, giả vờ chỉnh xương cổ. Tôi dừng lại, nhìn lơ láo, tỏ vẻ trầm ngâm. "Ô!" tôi nói, "tôi nghĩ là tôi biết một cách hay hơn!" Jerry đảo mắt và thở dài, đợi tôi bắt tay vào làm. Tôi quyết định không đi quá trớn, thế rồi tôi bắt đầu làm việc.

Để viết một hồ sơ xuyên qua socket, Jerry đã phải gởi ba dòng chữ trước. Một dòng có string "Sending", dòng kế tiếp chứa tên hồ sơ và dòng cuối chỉ định chiều dài của hồ sơ. Rồi sau đó Jerry mới gởi chính hồ sơ ấy như một chuỗi từ. Đoạn mã như sau:

```
private void writeSendFileCommand() throws IOException {
  os.println("Sending");
  os.println(itsFilename);
  os.println(itsFileLength);
  char buffer[] = new char[(int) itsFileLength];
```

```
  fileReader.read(buffer);

  os.write(buffer);

  os.flush();

}
```

Khi đọc hồ sơ ngược lại từ socket, gã gọi readLine ba lần, mỗi lần cho mỗi ba dòng gã gởi đi. Gã dùng "Sending" string như một phương thức nhận diện của một chuyển xuất và lưu giữ cái thứ nhì như tên của hồ sơ; cái thứ ba là chiều dài của hồ sơ. Gã dùng nó để chỉ định chuỗi từ được dùng như một tầng đệm. Rồi sau đó gã dùng chiều dài để đọc số lượng từ thích ứng từ socket.

```
private void parse(String cmd) throws Exception {

  if (cmd != null) {

    if (cmd.equals("Sending")) {

      filename = is.readLine();

      fileLength = Long.parseLong (is.readLine());

      content = new char[(int)fileLength];

      is.read(content,0, (int)fileLength);

      fileReceived = true;

    }

  }

}
```

Mọi thứ làm việc ngon lành, nhưng tôi biết cách hay hơn. Đầu tiên, tôi đổi đoạn test để đọc objects thay vì những dòng:

```
public void serve(Socket socket) {
```

```
  try {
    os = new PrintStream(socket.getOutputStream());
    is = new ObjectInputStream(socket.getInputStream());
    os.println("SMCR Test Server");
    os.flush();
    parse((String)is.readObject());
  }
  catch (Exception e) {
  }
}

private void parse(String cmd) throws Exception {
  if (cmd != null) {
    if (cmd.equals("Sending")) {
      filename = (String)is.readObject();
      fileLength = is.readLong();
      content = (char[]) is.readObject();
      fileReceived = true;
    }
  }
}
```

Kế tiếp tôi thay đổi phần SMCRemoteClient để viết objects thay vì strings.

```
public boolean connect() {
```

```
  ...
  os = new ObjectOutputStream(smcrSocket.getOutputStream());
  ...
}

private void writeSendFileCommand() throws IOException {
  os.writeObject("Sending");
  os.writeObject(itsFilename);
  os.writeLong(itsFileLength);
  char buffer[] = new char[(int) itsFileLength];
  fileReader.read(buffer);
  os.writeObject(buffer);
  os.flush();
}
```

Tôi chạy trọn bộ các tests và chúng làm việc ngon lành. "Thấy chưa?" tôi gáy. "Tôi nghiệm ra viết objects thay vì strings thì tốt hơn."

Eureka!

Tôi nhìn Jerry, nhưng có gì đó thay đổi - đôi mắt gã không tập trung. Gã đứng dậy và bắt đầu rảo quanh. Thỉnh thoảng gã dừng lại, nhìn vào màn hình, nhìn tôi, lắc đầu và lại tiếp tục rảo bước. Gã lẩm nhẩm gì đó về năm tháng, kinh nghiệm và sự ngu xuẩn. Tôi hơi hãi.

Sau rốt, gã dừng lại, nhìn tôi thẳng vào mắt và nói: "À, Alphonse, mày xong rồi đó."

"Tôi làm gì sai vậy Jerry?" tôi thì thầm.

Gã nhìn tôi chằm chặp vài giây. Thế rồi gã xoay người hướng về thang máy và ra lệnh, "đi theo tao."

Chuyến đi trên thang máy yên lặng như nhà mồ. Trạng thái của Jerry khó mà đoán nổi: gã không hẳn là giận dữ nhưng chắc chắn là gã bực dọc, và lẽ gì đó tôi đã dính vào sự bực dọc này. Trong thang máy, chúng tôi lặng lẽ thay đổi vị trí để giảm mức lệch coriolis -2- tôi cố nghiệm ra lý do tại sao phần mã nguồn đơn giản tôi thay đổi có thể tạo ảnh hưởng ghê gớm đến gã như thế.

Tôi theo Jerry vào một phòng khách ở một trong những tầng thuộc "low-g". Các tay học việc không thường được phép vào các tầng trên .49g. Trên đường đi lên, tôi không dõi các bảng hiệu của các tầng lầu nhưng tầng này có vẻ thấp hơn .4g. Bên trong phòng khách có năm gã "du hành" lập trình viên khác. Jerry giới thiệu tôi với nhóm này. Tôi gắng nhớ hết tên của mọi người: Johnson, Jasmine, Jason, Jasper và Jennifer. Jerry bảo tôi đứng giữa phòng khách trong khi gã và mọi người ngồi trên salon xung quanh tôi. Sau đó Jerry xoay về phía nhóm lập trình viên và với vẻ kiểu cách, gã tuyên bố, "À, có chuyện đã xảy ra. Tôi tin rằng Alphonse là tay học việc đầu tiên trong năm vào "Micah his Journeyman."

Tôi cảm thấy ngực tôi ngừng đập một nhịp và mắt tôi mở rộng ra. Đây là điều hết sức đơn giản! tôi không dự tưởng điều này!

"Có ai làm Micahed năm nay chưa nhỉ?" Jerry hỏi. Tiếng xì xầm lan ra khắp phòng nhưng mọi người đều lắc đầu - hiển nhiên là chưa có ai.

Jasmine nhìn tôi chằm chặp hồi lâu. Trong khi cô ta dán mắt vào tôi, nàng bảo Jerry: "OK, Jer, cho bọn tôi nghe câu chuyện ấy đi."

Jerry thở dài, gã cố gắng một cách rõ rệt để lấy lại tư thế và bắt đầu nói.

"Như các bạn biết, ông C yêu cầu tôi làm cái SMCRemote cho nó chạy." Đám lập trình viên đều gật đầu; hiển nhiên họ biết chuyện này. "Alphonse và tôi bỏ ra cả ngày cho bài tập SocketServer; và nó làm việc rất tốt."

Thêm một cú sốc: SocketServer chỉ là một bài tập?

"Từ lúc làm cho nó chạy được, chúng tôi bắt đầu đặt phần client của SMCRemote lại với nhau. Một trong những test cases là chuyển tải một hồ sơ từ client đến server qua socket." Lại thêm những cái gật đầu trong phòng.

Jerry càng bối rối thấy rõ. Gã trăn trở trên ghế và tránh những ánh mắt, gã nhìn chằm chặp xuống sàn nhà. "Tôi chỉnh định việc chuyển xuất hồ sơ bằng cách gởi ba dòng chữ theo sau bằng một chuỗi từ. Dòng đầu tiên là danh tính của việc chuyển xuất, dòng thứ hai là tên hồ sơ và dòng thứ ba là chiều dài hồ sơ." Lại thêm gật đầu - điều này chẳng làm họ ngạc nhiên tí nào.

"Rõ ràng, đây chỉ là một cách đơn giản cho mấy cái test có thể đạt để chúng tôi có thể refactor thành một dạng tốt hơn." Lại thêm gật đầu; thêm những tiếng xầm xì đồng ý. "Và rồi..." Jerry ngừng lại. "Alphonse nói là hắn nghĩ là hắn có à... ờ.... một ý kiến hay hơn."

Căn phòng trở nên yên tĩnh. Đôi mắt của Jasmine vẫn dán chặt vào tôi, nhưng cái nhìn của nàng chuyển từ trạng thái đánh giá sang suy đoán. Từng người một, tôi cảm thấy những tia nhìn của các tay "du hành" ngừng lại ở tôi. Làm gì mà lớn chuyện vậy? Tại sao họ đang đòi cái Micah cho tôi nhỉ?

Johnson là người phá tan không khí u ám.

"Không phải bồ muốn cho bọn tôi biết --" gã buộc miệng nói, rồi ghìm lại bằng một cú hít vào nặng nề.

Liếc nhìn, tôi thấy Jerry đang gật đầu. Gật đầu cho chuyện gì nhỉ?

Đề nghị ngây thơ

Tôi không chịu nổi nữa. Tôi rời khỏi cái nhìn của Jasmine, nhìn thẳng vào mắt của từng tay "du hành" trong phút chốc rồi nói: "Tất cả những gì tôi đề nghị chỉ là việc chuyển tải objects thay vì strings! tôi chẳng thấy việc ấy lại là một Micah!"

Jennifer bước về phía tôi và nói, "Vâng, bồ chỉ làm ngần ấy. Và, không, tôi không giả định là bồ nghĩ ngợi gì nhiều về nó - nhưng với bọn tôi, đây là chuyện lớn."

"Tại sao?" tôi rít lên, thật sự hoảng sợ.

"Bởi," Jerremy giải thích, "đặc điểm quan trọng nhất của một lập trình viên giỏi là khả năng suy nghĩ một cách trừu tượng. Thật ra rất ít người có thể làm như thế. Mày mới vừa chứng tỏ là mày có thể làm điều này."

Tôi đâm nghi ngờ. "Nó chỉ là một object thôi mà," tôi lặp bắp.

"Chính xác," Jennifer nói. Bọn họ đều gật đầu một cách nghiêm chỉnh.

Tôi lắc đầu. "Ôi, thì, nếu đây là điều hay - một Micah gì đó - tại sao Jerry có vẻ cáu kỉnh vậy?"

"Ô, chuyện ấy!" Jasmine cười to. "Jerry xuống đây vào giờ nghỉ lần trước và kể cho bọn mình về vấn đề chiều dài hồ sơ của cậu. Anh ấy chắc rằng cậu sẽ rất có ấn tượng khi thấy chiều dài của hồ sơ sẽ khớp khít vào chuyển xuất hồ sơ. Anh ấy dự phỏng cậu sẽ ngạc nhiên biết chừng nào."

"Ừa," Jasper cười điệu đàng, "và cậu lại đi mà chỉ cho gã chiều dài này trở nên lạc đề."

Tôi nuốt nước bọt, cố chịu đựng. "Tôi đã làm thế sao?"

Jerry đứng dậy và nói, "ngẫm lại chuyện đó đi Alphonse. Nếu mày gởi một chuỗi từ như một object, tại sao mày còn phải gởi chiều dài của hồ sơ riêng ra nữa? Trong sáu tháng tới đây, mấy tay này thế nào cũng sẽ nạo sườn tao về chuyện này" gã nói thêm một cách thiểu não.

"Bọn tớ chắc chắn sẽ làm thế!" Jennifer cười toe toét. "Mỗi khi xét duyệt mã nguồn của anh ấy, bọn tớ sẽ hỏi anh tham số chiều dài hồ sơ ở đâu!" Cô ta cười rúc rích trong khi Jerry nhăn nhó và cứng đờ khuôn mặt.

"Cậu phải biết, Alphonse," Jasmine giải thích, "không những cậu đã tạo nên một bước trừu tượng đáng kể, giải pháp của cậu còn đơn giản hơn giải pháp của Jerry. Hơn nữa, nó là một cách đơn thuần phế bỏ dự tính của Jerry với nhu cầu chiều dài của hồ sơ. Cậu đã Micahed anh ấy!"

Tôi bắt đầu hiểu ra sự thể. Ít ra tôi không bị dính vào một phiền toái nào...

"Tôi nghĩ là," Jasmine nói, "một biến cố như thế này cần đổi cặp (làm việc). Jerry, tôi đổi người học việc với anh. Anh nhận Andy và tôi sẽ làm việc với Alphonse vài ngày."

... hay là tôi?

-1- "Micah" ở đây, trong bài này, có lẽ là một loại đặc quyền hoặc một vinh dự lớn lao. Theo tự điển Merriam-Webster thì Micah là tên của một nhà tiên tri người Do Thái ở thế kỷ thứ 8 sau Công nguyên. "Micah" xuất xứ từ nguyên thủy chữ MIkhAyAh (tiếng Hebrew).

-2- Coriolis: tên của nhà toán học, kỹ sư công chánh người Pháp Gaspard G. Coriolis. Xem thêm tiểu sử và công nghiệp của Coriolis ở: http://www-gap.dcs.st-and.ac.uk/~history/Mathematicians/Coriolis.html

The Crafsman 14. Transaction Action

Hành động chuyển tải

Làm việc với một tay du mục -1- mới là một kinh nghiệm nặng nề cho Alphonse. Liệu chàng chịu nổi tia nhìn sắc bén của Jasmine và xứng đáng với cái tên lóng mới của mình?

Robert C. Martin

"OK, cao thủ -2- xem thử cậu thế nào." Độ căng thẳng trong cái nhìn của Jasmine làm tôi dán chặt vào ghế. "Ý... ý cô thế nào?" tôi lắp bắp. "Thôi đi cao thủ, bộ cậu định không làm tôi quê như cậu đã làm Jerry quê sao," nàng nói. "Tôi chẳng cố tình làm cho ai quê cả," tôi chống chế một cách yếu ớt. "Tôi chỉ...." "Ừa, hẳn nhiên rồi," nàng bất chợt ngắt ngang câu nói của tôi, tỏ vẻ cháng chường. "Thôi, hãy bắt tay vào công việc cho rồi. Cậu định sẽ thay đổi những gì tiếp theo đây?"

Chúng tôi ngồi trong phòng làm việc, xem xét đoạn mã Jerry và tôi vừa viết xong. Tôi chỉ cho Jasmine cách tôi thay đổi đoạn code của Jerry dùng để gởi strings thay vì objects xuyên qua socket.

"Tôi... ùm... ờ không biết. Tôi chỉ nghĩ là gởi objects chắc tốt hơn strings." Nàng làm tôi hết sức bối rối. Mức căng thẳng cứ đổ dồn từ ánh mắt và thái độ của nàng. "Suy nghĩ đi cao thủ, suy nghĩ! Cậu không chỉ thuần tuý gói vài cái strings và integers vào trong một object, phải thế không? Gói như vậy ngầm định vấn đề gì? Cậu có thể làm gì với nó?"

"Tôi, ùm...." giá như đôi mắt nàng ngưng đè nặng lên tôi, có lẽ tôi có thể suy gẫm. Tôi nhắm nghiền đôi mắt và thầm tụng một đoạn kinh -3-. Trong vòng vài giây, tôi đã có thể xem xét câu hỏi của nàng.

Cái test case chúng tôi đã làm dùng để xác thực chúng tôi có thể gởi hồ sơ có xuyên qua socket. Đoạn mã dùng để gởi hồ sơ như thế này:

```
private void writeSendFileCommand() throws IOException {

  os.writeObject("Sending");

  os.writeObject(itsFilename);

  os.writeLong(itsFileLength);

  char buffer[] = new char[(int) itsFileLength];

  fileReader.read(buffer);

  os.writeObject(buffer);

  os.flush();

}
```

Nhưng tại sao chúng tôi lại gởi hồ sơ đi? Chúng tôi gởi nó đến SMCRemoteServer để được biên dịch. Sau đó server sẽ trả về hồ sơ đã được biên dịch. Tại sao Jerry lại gởi "Sending" string trước? Gã nói rằng mục đích là để báo server có hồ sơ đang được gởi đến - nhưng chúng tôi lại không muốn thông báo cho server là có hồ sơ đang được gởi đến; chúng tôi muốn ra lệnh cho server biên dịch một hồ sơ và gởi ngược lại kết quả.

Tôi suy nghĩ rất kỹ lưỡng, nhưng một phần nào đó trong não bộ của tôi vẫn đang tiếp tục tụng kinh. Hầu như trong trạng thái nửa tỉnh, nửa mê, tôi đi thẳng đến bức tường và vẽ ra sơ đồ "kết quả biên dịch". Tôi nhác thấy khuôn mặt nghiêm trọng của Jasmine thoáng một nụ cười. "Tôi khoái cái lối suy nghĩ của cậu đó cao thủ. Đừng dừng lại ở đó."

Bốn mẩu dữ liệu được gởi đến server: tên hồ sơ, độ dài hồ sơ, nội dung hồ sơ và chuỗi "Sending". Tại sao những mẩu này được gởi riêng biệt? Chúng đều thuộc vào một gói tin của một xuất chuyển tải! Đúng rồi! Tôi tự lắc đầu với chính mình và thay đổi đoạn test như sau:

```
public void testCompileFile() throws Exception {

  File f = createTestFile("testSendFile", "I am sending this file.");

  c.setFilename("testSendFile");

  assertTrue(c.connect());
```

```
  assertTrue(c.prepareFile());
  assertTrue(c.compileFile());
  Thread.sleep(50);
  assertTrue(server.fileReceived);
  assertEquals("testSendFile", server.filename);
  assertEquals(23, server.fileLength);
  assertEquals("I am sending this file.", new String(server.content));
  f.delete();
}
```

Thế rồi tôi đổi hàm sendFile cũ như sau:

```
public boolean compileFile() {
  boolean fileSent = false;
  char buffer[] = new char[(int) itsFileLength];
  try {
    fileReader.read(buffer);
    CompileFileTransaction cft =
      new CompileFileTransaction(itsFilename, buffer);
    os.writeObject(cft);
    os.flush();
    fileSent = true;
  }
  catch (Exception e) {
    fileSent = false;
```

```
  }

  return fileSent;

}
```

Jasmine theo dõi rất sát sao. Tôi không thể dò nổi cảm giác của nàng nhưng tôi biết chắc là mình đang đi đúng hướng. Kế tiếp tôi viết CompileFileTransaction class:

```
public class CompileFileTransaction implements Serializable {

  private String filename;

  private char contents[];

  public CompileFileTransaction(String filename, char buffer[]) {

    this.filename = filename;

    this.contents=buffer;

  }

  public String getFilename() {

    return filename;

  }

  public char[] getContents() {

    return contents;

  }

}
```

Đoạn này cho phép chương trình được biên dịch. Tất nhiên là mấy cái test bị hỏng, bởi thế tôi lại thay đổi phần server giả như sau:

```
public void serve(Socket socket) {
  try {
    os = new PrintStream(socket.getOutputStream());
    is = new ObjectInputStream(socket.getInputStream());
    os.println("SMCR Test Server");
    os.flush();
    parse(is.readObject());
  }
  catch (Exception e) { }
}
private void parse(Object cmd) throws Exception {
  if (cmd != null) {
    if (cmd instanceof CompileFileTransaction) {
      CompileFileTransaction cft = (CompileFileTransaction) cmd;
      filename = cft.getFilename();
      content = cft.getContents();
      fileLength = content.length;
      fileReceived = true;
    }
  }
}
```

Những thay đổi này giúp cho các phần test đều đạt. "Phải ý cô giống như thế này không?" Tôi hỏi. "Ưa, chỉ là khởi điểm thôi," nàng xác nhận một cách dè chừng. "Chắc chắn là nó hay hơn lối chuyển mỗi phần dữ liệu thành strings của Jerry - và nó cũng hay hơn lối chuyển gởi mỗi phần dữ liệu riêng biệt." "Vậy cô làm thế nào cho hay hơn nữa vậy?" Tôi hỏi. "Hẵn đã," nàng nói một cách thiếu kiên nhẫn. "Ngay lúc này hãy hoàn tất phần chuyển tải. Cậu phải làm cho client tiếp nhận hồi đáp từ server." "Cái đó chắc không khó lắm," tôi đáp, cảm thấy phấn chấn hơn một chút, và thêm vào ba dòng như sau vào đoạn testCompileFile như sau:

```
File resultFile = new File("resultFile.java");

assertTrue("Result file does not exist", resultFile.exists());

resultFile.delete();
```

Tôi chạy đoạn test và xác thật nó bị hỏng. "Sau khi mình gọi compileFile, kết quả hẳn phải được viết vào một hồ sơ," tôi giải thích cho Jasmine, rồi nói thêm, "ngay lúc này tôi không quan tâm đến chuyện có gì trong hồ sơ; tôi chỉ muốn chắc là hồ sơ đó được tạo ra." "Vậy cậu làm cách nào để tạo ra nó?" nàng thách thức. "Tôi sẽ cho cô thấy," tôi nói, thay đổi đoạn server giả như sau:

```
private void parse(Object cmd) throws Exception {
  if (cmd != null) {
    if (cmd instanceof CompileFileTransaction) {
      CompileFileTransaction cft = (CompileFileTransaction) cmd;
      filename = cft.getFilename();
      content = cft.getContents();
      fileLength = content.length;
      fileReceived = true;
      CompilerResultsTransaction crt =
        new CompilerResultsTransaction("resultFile.java");
      os.writeObject(crt);
      os.flush();
```

```
    }
  }
}
```

Thế rồi tôi tạo phần biên dịch này bằng cách thêm một cái sườn của CompileResultsTransaction class

```
public class CompilerResultsTransaction implements Serializable {
  public CompilerResultsTransaction(String filename) { }
  public void write() { }
}
```

Tất nhiên phần test vẫn hỏng, bởi thế tôi thay đổi compileFile như sau:

```
public boolean compileFile() {
  boolean fileCompiled = false;
  char buffer[] = new char[(int) itsFileLength];
  try {
    fileReader.read(buffer);
    CompileFileTransaction cft =
      new CompileFileTransaction(itsFilename, buffer);
    os.writeObject(cft);
    os.flush();
    Object response = is.readObject();
    CompilerResultsTransaction crt = (CompilerResultsTransaction)response;
```

```
    crt.write();
    fileCompiled = true;
   }
   catch (Exception e) {
    fileCompiled = false;
   }
   return fileCompiled;
  }
```

Cuối cùng, tôi thực hiện chi tiết phần chuyển tải:

```
public class CompilerResultsTransaction implements Serializable {
  private String filename;
  public CompilerResultsTransaction(String filename) {
    this.filename = filename;
  }

  public void write() throws Exception {
    File resultFile = new File(filename);
    resultFile.createNewFile();
  }
}
```

Kết quả biên dịch

Tại sao tên, độ dài, nội dung của hồ sơ và "Sending" string đều được gởi đến server riêng biệt nếu chúng đều thuộc về một chặng chuyển tải?

"Ở giai đoạn này được vậy là tốt rồi," Jasmine nói. "Tôi đi giải lao một chốc trong khi cậu thực hiện xong quy trình CompileResultsTransaction thực sự viết thành hồ sơ thay vì chỉ tạo ra nó. Cũng nên dọn dẹp chút đỉnh nữa. Có khá nhiều mảnh vụn vặt còn sót lại trong lúc cậu và Jerry khuấy vọc chuyện gởi strings và integers. Nhưng trước khi tôi đi, tôi muốn biết ý kiến của cậu trong phần instanceof cậu dùng trong đoạn server giả."

Đó là giải pháp đơn giản nhất mà thôi có thể nghĩ ra dùng để kiểm tra xem object sắp trả lại có thật sự là CompileFileTransaction hay không." Tôi nói, bắt đầu cảm thấy bối rối. "Có gì sai với phần này sao?"

Nàng đứng lên, nhìn về phía tôi và trả lời, "không có gì sai trầm trọng, nhưng cậu có nghĩ rằng server thật sẽ làm thế sao? Liệu server thật sẽ có chuỗi if/else dài ngoẵng cho instanceof để mà biến xuất các chuyển tải đi vào?"

"Tôi chưa nghĩ xa đến như thế," tôi thú nhận. "Không," nàng nói một cách thẳng thừng, "Tôi không hình dung cậu nghĩ xa như vậy." Và rồi nàng rảo bước ra khỏi phòng.

Căn phòng trống rỗng khi không có nàng, như thể sự hiện diện của tôi chẳng có giá trị gì. Tôi thở dài và ngúc ngoắc cái đầu. Làm việc với Jasmine sắp tới sẽ đầy mệt mỏi và đầy sự giáo huấn đủ mọi kiểu. Một điều tôi biết chắc - tôi ghét bị gọi là cao thủ. Tôi lại thở dài và bắt đầu giải quyết công tác nàng giao cho.

-1- Dựa trên góp ý của cl trong bài thứ 13 , journeyman có thể được xem như những kẻ du mục, đi tìm những vùng "đất mới". Nghĩa bóng cho journeyman cũng hết sức thích hợp cho những lập trình viên có cái nhìn khai phá. Tôi tạm dịch journeyman là du mục theo tinh thần này.

-2- Hotshot: tiếng lóng chỉ cho một cá nhân kinh nghiệm và nổi bật. Hotshot cũng có thể dùng với tính cách châm biếm, bỡn cợt hoặc thân thiện. Trong bài này, có lẽ Jasmine gọi Alphonse với tính cách bỡn cợt.

-3- Mantra: có nghĩa chung là đoạn kinh kệ. Theo đạo Hindu và đạo Phật, mantra có khả năng hoá giải những trắc trở.

The Crafsman 15. Ess Are Pee

"Ếch" là "Bê" -1-

Từ chuyện tay học việc nhiệt tình của chúng ta dọn dẹp hồ sơ Jasmine yêu cầu, dẫn đến tình trạng quá thái trong lúc anh chàng hình dung một cuộc đối thoại tưởng tượng - với chính anh ta.

Robert C. Martin

Hình ảnh cuối tôi thấy trước khi cánh cửa khép lại là mái tóc dài óng mượt của Jasmine, vung vẩy theo nhịp bước đầy chủ ý của nàng. Khi căn phòng tan lắng bóng dáng nàng , tôi cảm thấy lồng ngực của mình nhẹ nhõm trút đi một luồng khí nén chặt. Đôi mắt tôi mất đi tiêu điểm, và suốt nhiều phút gần như tỉnh táo, tôi ngồi thừ, nhìn về cánh cửa mờ nhoà đi trong tầm mắt.

Vẫn còn đờ người ra, tôi nhận ra mình phải trở lại làm việc - nhưng làm thế nào đây? Không biết nếu Jasmine làm, nàng sẽ viết hồ sơ và dọn dẹp đoạn mã ra sao nhỉ? Tôi biết chắc là nàng sẽ nói là tôi chẳng có gì xuất sắc cho nàng xem khi nàng trở lại: "Cao thủ mà hoá ra như vậy sao! Nãy giờ cậu chỉ ngồi thừ ra đó vọc khuấy mấy ngón tay cái phải không?"

"OK, Jasmine, OK," Tôi nói. "Mình làm gì trước đây?"

"Nào, cao thủ, chúng mình phải làm sao cho CompilerResultsTransaction chuyên ch ở nội dung của một hồ sơ từ server đến client, và rồi viết hồ sơ ấy xuống client."

"Ồ, đúng rồi", tôi đáp. "Trình dịch sẽ tạo ra một hồ sơ xuất trên server, và chúng ta phải dời nó đến client - y như thể chúng ta vừa thực hiện trong CompileFileTransaction."

```
public class CompileFileTransaction implements Serializable {

 private String filename;

 private char contents[];

 public CompileFileTransaction(String filename, char buffer[]) {

  this.filename = filename;

  this.contents=buffer;

 }

 public String getFilename() {

  return filename;

 }

 public char[] getContents() {

  return contents;

 }

}
```

"Chúng ta mở và đọc hồ sơ trong compileFile method, rồi tạo và chuyển tên hồ sơ và chuỗi ký tự vào constructor của CompileFileTransaction." Tôi tiếp tục.

```
public boolean compileFile() {

 char buffer[] = new char[(int) itsFileLength];
```

```
try {
  fileReader.read(buffer);
  CompileFileTransaction cft =
    new CompileFileTransaction(itsFilename, buffer);
```

"Rồi, cao thủ, mình vừa làm đúng y như vậy. Có điểm nào cậu không vừa lòng chăng?"

Tôi không muốn viết đoạn mã y hệt như nhau hai lần. Jerry phản đối cực lực chuyện lặp lại mã nguồn.

"Jerry không phải là con dao bén nhất trong tủ đâu -2-, cao thủ."

"Tôi không rõ chuyện đó nhưng tôi nghĩ, với quan điểm này thì anh ta đúng. Bởi thế, tôi nghĩ là tôi muốn viết một class dùng để mang hồ sơ xuyên qua socket."

"Cái gì đó tương tự như FileCarrier?"

"Ừa, cái tên đó hay á!"

"Rồi, cao nhân -3-, viết một cái test cho nó đi."

"Cao nhân? - èm, OK. Thế này nhé?"

```
public class FileCarrierTest extends TestCase {
  public void testAFile() throws Exception {
    final String TESTFILE = "testFile.txt";
```

```
    final String TESTSTRING = "test";
    createFile(TESTFILE, TESTSTRING);
    FileCarrier fc = new FileCarrier(TESTFILE);
    fc.write();
    assertTrue(new File(TESTFILE).exists());
    String contents = readFile(TESTFILE);
    assertEquals(TESTSTRING, contents);
  }
}
```

"Hay lắm, ông mãnh. Tiếp tục đi."

"Được rồi, thưa cô J. Sau đây là hai function tiện ích..."

```
private String readFile(final String TESTFILE) throws IOException {
  BufferedReader reader = new BufferedReader(new FileReader(TESTFILE));
  String line = reader.readLine();
  return line;
}

private void createFile(final String name, final String content)
            throws IOException {
  PrintWriter writer =
    new PrintWriter(new PrintWriter(new File Writer(name));
  writer.println(content);
```

```
  writer.close();
}
```

"... và đây là phần ứng dụng rút gọn của FileCarrier sẽ làm cho phần test biên dịch và không chạy khi test."

```
public class FileCarrier {
  public FileCarrier(String fileName) { }
  public void write() { }
}
```

"Rồi, quá đã -4-, mình chỉ cần làm cho cái test đạt bằng cách đọc hồ sơ trong constructor và viết nó trong function write."

"Chưa đâu, ông tướng - đầu tiên là chạy cái test và biết chắc nó hỏng cái đã."

"Ơ, nàng J, chưa có ứng dụng FileCarrier. Tất nhiên là nó sẽ hỏng thôi."

"Hả, hoả quân cậu và tôi biết như vậy. Nhưng liệu chương trình có biết thế không?"

"Tôi thật là khoái những khi cô muốn tôi chạy test. OK, này." [Phần test đạt] "Hở? làm sao có thể như vậy được?"

"Quỷ tha, tôi không rõ nữa. Làm sao phần test có thể đạt trong khi chẳng có gì trong FileCarrier đến --"

"Egad, Auriculatum! -5- mình chưa hề xoá hồ sơ test."

"À, ông kẹ. Thế sao cậu không chữa nó đi?"

"OK, đây."

```
public void testAFile() throws Exception {
  final String TESTFILE = "testFile.txt";
  final String TESTSTRING = "test";
  createFile(TESTFILE, TESTSTRING);
  FileCarrier fc = new FileCarrier(TESTFILE);
  new File(TESTFILE).delete();
  c.write();
  assertTrue(new File(TESTFILE).exists());
  String contents = readFile(TESTFILE);
  assertEquals(TESTSTRING, contents);
}
```

"Ngon lành, bây giờ nó hỏng rồi! nhưng mà ông thần, cậu làm cho nó đạt được không?

"Hiển nhiên rồi, JJ - xem đây!"

```
public class FileCarrier implements Serializable {
  private String fileName;
  private char[] contents;
```

```
  public FileCarrier(String fileName) throws Exception {
    File f = new File(fileName);
    this.fileName = fileName;
    int fileSize = (int)f.length();
    contents = new char[fileSize];
    FileReader reader = new FileReader(f);
    reader.read(contents);
    reader.close();
  }

  public void write() throws Exception {
    FileWriter writer = new FileWriter(fileName);
    writer.write(contents);
    writer.close();
  }
}
```

"Yee Hah! đại nhân, bây giờ cậu mới nên cơm nên cháo đây -6-!"

"Không có chi, cám ơn đại nương -7-, nhưng cô cũng chưa thấy gì mà. Hãy xem lối tôi tích hợp FileCarrier vào CompileFileTransaction -- cái này chắc sẽ làm cô chú ý."

```
public class CompileFileTransaction implements Serializable {
  FileCarrier sourceFile;
```

```
  public CompileFileTransaction(String filename)  throws Exception {
    sourceFile = new FileCarrier(filename);
  }
  public String getFilename() {
    return sourceFile.getFileName();
  }
  public char[] getContents() {
    return sourceFile.getContents();
  }
}
```

"Và bây giờ tôi sẽ đổi function compileFile để dùng cái CompileFileTransaction mới đây!"

```
CompileFileTransaction cft = new CompileFileTransaction(itsFilename);
os.writeObject(cft);
os.flush();
Object response = is.readObject();
CompilerResultsTransaction crt = (CompilerResultsTransaction)response;
crt.write();
```

"Và bây giờ tôi chạy mấy cái test và.... thấy chưa? Chúng đạt hết!"

"Ô, đại cao thủ, tuyệt! Cậu đã xuất ra dăm ba tuyệt chiêu!"

"Tính trước hết rồi mà, Dạ Hương -8-, tính hết rồi. Bây giờ hãy xem tôi đặt cái FileCarrier và trong CompilerResultsTransaction !"

```
public class CompilerResultsTransaction implements Serializable {

  private FileCarrier resultFile;


  public CompilerResultsTransaction(String filename)  throws Exception {

    resultFile = new FileCarrier(filename);

  }


  public void write() throws Exception {

    resultFile.write();

  }

}
```

"Ôi chao!"

"Và hãy xem cách tôi đổi cái test dùng trong transaction mới một cách nhà nghề đây!"

```
private void parse(Object cmd) throws Exception {

  if (cmd != null) {

    if (cmd instanceof CompileFileTransaction) {

      CompileFileTransaction cft = (CompileFileTransaction) cmd;

      filename = cft.getFilename();

      content = cft.getContents();
```

```
      fileLength = content.length;
      fileReceived = true;

      TestSMCRemoteClient.createTestFile("resultFile.java", "Some content.");

      CompilerResultsTransaction crt =
        new CompilerResultsTransaction("resultFile.java");
      os.writeObject(crt);
      os.flush();
    }
  }
}
```

"Làm sao cậu biết được hết mấy thứ này vậy, Alphonse?"

"Cô thấy đó, Jasmine, tôi biết hết mọi chuyện thuộc về objects. FileCarrier là một object đó Jasmine. Cô có thấy nó có thể được xử dụng nhiều hơn chỉ một nơi không? Cô có thấy một object chỉ hàm chứa một trách nhiệm? Cô thấy không? Cô có biết Nguyên Lý Trách Nhiệm Đơn -9- không Jasmine? Có khi nào cô nghe đến nó chưa? đã nghiên cứu nó chưa? Tôi nghiên cứu nó rồi đó. Cô biết nó nói sao không, Jasmine? ***Nó nói rằng một class chỉ nên có một và chỉ một lý do để thay đổi. Nó nói rằng mọi functions và variables của một class chỉ nên làm việc để cùng đưa đến một mục đích. Một class không nên cố gắng hoàn thành nhiều hơn một mục đích***.... Cô có đang lắng nghe đó không, Jasmine?"

"Vâng, Alphonse. Tôi đang chăm chú mà."

"Tiếp tục lắng nghe, Jasmine. Trong đám bọn mình ai mà biết nguyên lý này gọi nó là SRP -10-. Đó là ESS ARE PEE Jasmine."

"ESS ARE PEE, Alphonse. Ess are pee."

"Còn nhớ function compileFile không? Còn nhớ cách nó dùng để đọc hồ sơ và chuyển chuỗi chữ cái vào trong CompileFileTransaction không? Cô hỏi tôi chuyện tôi không thích cái gì trong function đó - tôi sẽ nói cho cô hay tôi không thích cái gì: nó vi phạm nguyên lý SRP! Nó có hai lý do để thay đổi thay vì một. Nó phụ thuộc vào cả chi tiết cách đọc hồ sơ lẫn chế độ xây dựng và gởi transactions. Làm như vậy quá nhiều trách nhiệm, Jasmine hỡi - quá sức nhiều."

"Cậu làm tôi hoảng lên đây, Alphonse - Ôi! Alphonse!"

Ngay lúc ấy, cùng một lúc hàng loạt sự kiện xảy ra. Tôi thấy cánh cửa đã mở ra. Tôi nhận ra chiếc ghế bên cạnh trống rỗng. Tôi nghe tiếng vọng của giọng tôi nhại Jasmine còn dội lại từ mấy bức tường. Và, hơn hết, tôi thấy Jasmine đang đứng ngay cửa ra vào.

Đôi mắt nàng lạnh như tiền.

còn tiếp.... không biết chừng.

-1- Nguyên bản tiếng Anh tác giả chơi chữ SRP thành ESS ARE PEE. Cụm này hnd không biết phải dịch ra sao cho ổn nên dịch trại thành "ếch là bê" cho dí dỏm. Nếu có bạn nào có ý kiến nào hay, xin đóng ý.

-2- "the sharpest knife is the drawer", một ngạn ngữ ám chỉ cho một cá nhân nào đó tài giỏi nhất trong nhóm hoặc một việc gì đó tốt nhất trong hoàn cảnh cho phép.

-3- Các từ lóng "hottie", "hot stuff", "over-temp", exotherm", "boiling point", "tepid breath", "latent heat", "electron volt", "fever man" dùng trong bài có chủ ý gia tăng cường độ. Những từ này đều dùng để đề cao một cá nhân một cách dí dỏm, thân mật và chút gì đó chế diễu. hnd tạm dịch những từ này là "cao nhân", "ông mãnh", "ông tướng", "hoả quân", "ông kẹ", "ông thần", "đại nhân", "đại cao thủ" và biết chắc là không thể tìm các từ hóm hỉnh tiếng Việt tương tự để chuyển dịch cho mỗi chữ lóng tiếng Anh này.

-4- "Jazzy-wazzy" cũng là một cụm từ lóng chỉ cho sự ngon lành, nhuần nhuyễn, vừa ý. hnd tạm dịch là "quá đã" để bình dân hoá từ lóng này.

-5- "Egad, Auriculatum": Egad là một cách gọi cảm thán tương tự như "oh God!" và Auriculatum có nguồn gốc từ tiếng Latin, chỉ cho bộ "nghe" hoặc miêu tả hình dạng giống như chiếc lá, hoặc vành tai. Ngoài ra, "auricula" còn một số nghĩa bóng khác. Cụm "Egad, Auriculatum" này có thể dịch nôm na là "ôi trời, nghe đây" nhưng hnd để nguyên văn cho thêm phần.... bùa chú ;)

-6- Cụm "you are cooking" là một idiom rất phổ biến, chỉ cho sự tiến triển đúng hướng và tốt đẹp.

-7- "Grandiflorum" từ grandiflora tiếng La tinh chỉ cho giống hoa hồng mọc theo dạng bụi, khóm. Từ này ám chỉ cho nữ giới. Ở trên JJ gọi Alphonse là đại nhân nên bên dưới hnd tạm dịch theo là đại nương cho khớp với tinh thần.

-8- "Night Bloomer" chỉ cho các giống hoa nở ban đêm nên tạm dịch là Dạ Hương.

-9- "Single Responsibility Principle" tạm dịch là nguyên lý trách nhiệm đơn.

-10- "SRP" hay Single Responsibility Principle, xin đọc thêm tài liệu tiếng Anh ở: http://www.objectmentor.com/resource...icles/srp.

The Crafsman 16. Excess Politesse

Quá mức bặt thiệp

Khi chàng đã nhũn xuống vì hổ thẹn, một đô nặng cân về chuyện thái độ giúp Alphonse hoàn thành mã nguồn - và nàng Jasmine mới này làm chàng cực kỳ khó chịu.

Robert C. Martin

Jasmine đứng đó, nhìn tôi chằm chặp. Sau một phút im lặng căng thẳng, nàng đảo mắt, lắc đầu và rảo thẳng đến tôi. "Alphonse", nàng nói một cách nghiêm khắc, "đừng bao giờ tái diễn trò đó nữa." Quá xấu hổ, tôi gật đầu và nói, "vâng, Jasmine."

"Và từ rày về sau, gọi tôi là cô J." "Vâng.... cô J," tôi chống chế. Bằng cái khịt mũi khô khan, nàng nói, "hãy xem thử cậu đã làm những gì." Tôi chỉ cho nàng xem đoạn mã FileCarrier và các đoạn tests. Lúc đầu nàng có vẻ thoả mãn nhưng rồi nàng nói, "FileCarrier đọc hồ sơ bằng một cú đọc đơn và viết bằng một cú viết đơn."

```
public class FileCarrier implements Serializable {

 private String fileName;

 private char[] contents;



 public FileCarrier(String fileName) throws Exception {

  File f = new File(fileName);

  this.fileName = fileName;

  int fileSize = (int)f.length();

  contents = new char[fileSize];

  FileReader reader = new FileReader(f);

  reader.read(contents);

  reader.close();

 }



 public void write() throws Exception {

  FileWriter writer = new FileWriter(fileName);

  writer.write(contents);

  writer.close();

 }
```

```
  public String getFileName() {
    return fileName;
  }

  public char[] getContents() {
    return contents;
  }
}
```

"Phần này có thể làm việc cho các ví dụ nhỏ," nàng tiếp tục, "nhưng tôi chẳng dám chắc phần đọc sẽ không kết thúc sớm, và nó chỉ tràm một phần của chuỗi. Hơn nữa, hồ sơ chuyên chở qua socket đến một hệ thống khác, hệ thống này không biết chừng đang dùng một loại ký tự kết thúc dòng kiểu khác. Bởi thế, tôi không nghĩ FileCarrier sẽ làm việc ngon lành xuyên qua các hệ thống bên ngoài. Mình nên làm gì đây Alphonse?"

Tôi không sót một mảy. "À... ờ... cô J, có lẽ chúng ta nên đọc và viết các hồ sơ mỗi lần một dòng và chuyên chở những hồ sơ này theo danh sách các dòng."

"Được rồi, Alphonse. Cậu thay đổi nó như vậy đi." Từng chút một, tôi thay đổi FileCarrier. Tôi đặc biệt cẩn thận với việc làm các tests có thể chạy. Khi mọi thứ đâu vào đó, tôi refactor class này cho nó đọc và viết rõ ràng và sạch sẽ ở mức tối đa.

```
public class FileCarrier implements Serializable {
  private String fileName;
  private LinkedList lines = new LinkedList();

  public FileCarrier(String fileName) throws Exception {
    this.fileName = fileName;
```

```
  loadLines();
}

private void loadLines() throws IOException {
  BufferedReader br = makeBufferedReader();
  String line;
  while ((line = br.readLine()) != null)
    lines.add(line);
  br.close();
}

private BufferedReader makeBufferedReader() throws FileNotFoundException {
  return new BufferedReader(
    new InputStreamReader(
      new FileInputStream(fileName)));
}

public void write() throws Exception {
  PrintStream ps = makePrintStream();
  for (Iterator i = lines.iterator(); i.hasNext();)
    ps.println((String) i.next());
  ps.close();
}

private PrintStream makePrintStream() throws FileNotFoundException {
```

```
    return new PrintStream(new FileOutputStream(fileName));
  }

  public String getFileName() {
    return fileName;
  }
}
```

"Hay lắm Alphonse," nàng nói. "Nhưng tôi không nghĩ FileCarrierTest thực sự bảo đảm FileCarrier tái lập hồ sơ một cách cẩn mẫn. Tôi muốn xem thêm vài cái tests."

Lúc này nàng hết sức bặt thiệp. Một lần nữa, tôi dựng đoạn mã từng phần một, giữ cho các tests vẫn chạy được trong khi thay đổi mã nguồn. Tôi refactor cho đến khi mã nguồn sạch và rõ ràng tới mức tối đa tôi có thể làm được. Tôi không muốn tạo thêm bất cứ lý do nào làm cho nàng nổi cáu nữa.

```
public class FileCarrierTest extends TestCase {
  public void testFileCarrier() throws Exception {
    final String ORIGINAL_FILENAME = "testFileCarrier.txt";
    final String RENAMED_FILENAME = "testFileCarrierRenamed.txt";
    File originalFile = new File(ORIGINAL_FILENAME);
    File renamedOriginal = new File(RENAMED_FILENAME);

    ensureFileIsRemoved(originalFile);
    ensureFileIsRemoved(renamedOriginal);

    createTestFile(originalFile);
    FileCarrier fc = new FileCarrier(ORIGINAL_FILENAME);
```

```
  rename(originalFile, renamedOriginal);
  fc.write();

  assertTrue(originalFile.exists());
  assertTrue(filesAreTheSame(originalFile, renamedOriginal));

  originalFile.delete();
  renamedOriginal.delete();
}

private void rename(File oldFile, File newFile) {
  oldFile.renameTo(newFile);
  assertTrue(oldFile.exists() == false);
  assertTrue(newFile.exists());
}

private void createTestFile(File file) throws IOException {
  PrintWriter w = new PrintWriter(new FileWriter(file));
  w.println("line one");
  w.println("line two");
  w.println("line three");
  w.close();
}

private void ensureFileIsRemoved(File file) {
```

```
    if (file.exists()) file.delete();
    assertTrue(file.exists() == false);
  }

  private boolean filesAreTheSame(File f1, File f2) throws Exception {
    FileInputStream r1 = new FileInputStream(f1);
    FileInputStream r2 = new FileInputStream(f2);
    try {
      int c;
      while ((c = r1.read()) != -1) {
        if (r2.read() != c) {
          return false;
        }
      }
      if (r2.read() != -1)
        return false;
      else
        return true;
    }
    finally {
      r1.close();
      r2.close();
    }
  }
}
```

Nàng thẩm tra mã nguồn trong khi tôi viết và không hề nhìn tôi - ngay cả một lần. Khả năng tập trung và phán các câu nhận định của nàng còn hơn hẳn thái độ lạnh lùng kiểu cách của nàng.

"Tốt lắm, mã nguồn sạch đó Alphonse. Tôi thích cách cậu bảo đảm hồ sơ nguyên thuỷ được đặt tên lại và hồ sơ mới được tạo ra. Không có điều gì có thể nghi ngờ rằng FileCarrier tạo hồ sơ ở đây. Cũng không có cách nào hồ sơ cũ bị bỏ rơi. Nhưng tôi chưa thấy method filesAreTheSame bị hỏng. Cậu có nghĩ là nó thực sự làm việc đâu vào đó không?"

Tôi chẳng thấy một tí sơ sót nào trong mã nguồn, nhưng tôi không có ý định kiểm chứng trong khi thiếu bằng chứng. Bởi thế tôi bắt đầu viết vài cái tests cho method fileAreTheSame. Đầu tiên, tôi viết một cái test chứng minh method này làm việc ngon lành cho hai hồ sơ như nhau. Rồi tôi viết một cái test khác chứng minh hai hồ sơ khác nhau không mang lại kết quả so sánh bằng nhau. Tôi viết tiếp thêm một cái test nữa để chứng minh rằng nếu hồ sơ này là tiền hiệu (prefix) của hồ sơ kia thì sẽ không thể so sánh.

Kết cục tôi viết tổng cộng năm trường hợp test khác nhau và chúng có cả lô mã trùng lặp: Mỗi test viết hai hồ sơ. Mỗi test so sánh hai hồ sơ. Mỗi test xoá hai hồ sơ. Để loại trừ phần trùng hợp này, tôi dùng mẫu thiết kế Template Method. Tôi dời trọn bộ các phần mã chung vào trong một abstract class nền gọi là FileComparator, rồi dời trọn bộ các phần mã khác biệt thành dạng vô danh (anonymous). Và thế là mỗi trường hợp thử nghiệm tạo một phó bản để dùng không gì hơn ngoài nội dung của hai hồ sơ và tinh thần của giai đoạn so sánh.

```
public class FileCarrierTest extends TestCase {


  private abstract class FileComparator {

    abstract void writeFirstFile(PrintWriter w);

    abstract void writeSecondFile(PrintWriter w);


    void compare(boolean expected) throws Exception {

      File f1 = new File("f1");

      File f2 = new File("f2");

      PrintWriter w1 = new PrintWriter(new FileWriter(f1));
```

```
    PrintWriter w2 = new PrintWriter(new FileWriter(f2));
    writeFirstFile(w1);
    writeSecondFile(w2);
    w1.close();
    w2.close();
    assertEquals("(f1,f2)", expected, filesAreTheS ame(f1, f2));
    assertEquals("(f2,f1)", expected, filesAreTheSame(f2, f1));
    f1.delete();
    f2.delete();
  }
}

public void testOneFileLongerThanTheOther() throws Exception {
  FileComparator c = new FileComparator() {
    void writeFirstFile(PrintWriter w) {
      w.println("hi there");
    }

    void writeSecondFile(PrintWriter w) {
      w.println("hi there you");
    }
  };
  c.compare(false);
}
```

```
public void testFilesAreDifferentInTheMiddle() throws Exception {
  FileComparator c = new FileComparator() {
    void writeFirstFile(PrintWriter w) {
      w.println("hi there");
    }

    void writeSecondFile(PrintWriter w) {
      w.println("hi their");
    }
  };
  c.compare(false);
}

public void testSecondLineDifferent() throws Exception {
  FileComparator c = new FileComparator() {
    void writeFirstFile(PrintWriter w) {
      w.println("hi there");
      w.println("This is fun");
    }

    void writeSecondFile(PrintWriter w) {
      w.println("hi there");
      w.println("This isn't fun");
    }
  };
```

```
  c.compare(false);
}

public void testFilesSame() throws Exception {
  FileComparator c = new FileComparator() {
    void writeFirstFile(PrintWriter w) {
      w.println("hi there");
    }

    void writeSecondFile(PrintWriter w) {
      w.println("hi there");
    }
  };
  c.compare(true);
}

public void testMultipleLinesSame() throws Exception {
  FileComparator c = new FileComparator() {
    void writeFirstFile(PrintWriter w) {
      w.println("hi there");
      w.println("this is fun");
      w.println("Lots of fun");
    }

    void writeSecondFile(PrintWriter w) {
```

```
        w.println("hi there");
        w.println("this is fun");
        w.println("Lots of fun");
      }
    };
    c.compare(true);
  }
}
```

"Alphonse, quá tuyệt." Chuẩn y chính thức của nàng thật khác xa thái độ lạnh lùng thường lệ làm tôi cứ muốn gào lên.

"Tôi thích cách cậu xử dụng mẫu thiết kế Template Method để loại trừ sự trùng lặp. Nhiều tay học việc không học các mẫu thiết kế cho đến khi họ bị ép phải học. Tôi cũng thích cách cậu thử nghiệm phần so sánh nội tương (communitavity of equality). Mọi phần so sánh đều xảy ra song phương. Tuyệt hảo!"

"Cám ơn cô J," tôi lí nhí, thở phào nhẹ nhõm khi biết chắc mình không làm hư sự lần này.

The Crafsman 17. Call The Guard

Gọi bảo kê

*Dự kiến chuyện tệ hại nhất sau khi "đụng" với Jasmine, Alphonse học được cách dùng mới cho điều kiện cách **if** - và khám phá bên trong ngoại diện bình thường của người hướng dẫn mới hàm chứa một cái đầu tỉ mỉ - chương 17*

Robert C. Martin

Nhật ký thân mến: Tuần học việc đầu tiên của tôi với ông C đã hoàn tất. Tôi học được thật nhiều nhưng tôi cũng đã làm rối lung tung cả lên. Hồi thứ Hai, Jerry yêu cầu tôi viết một chương trình để tạo số nguyên. Đến thứ Ba, gã lại yêu cầu tôi viết chương trình tạo số nguyên tố. Trọn ngày thứ Tư dành cho việc làm SocketServer có thể chạy được. Thứ Năm chúng tôi bắt đầu làm việc với SMCRemoteClient và tôi đã Micahed -1- Jerry. Tôi gặp Jasmine chiều hôm ấy trong phòng khách của các tay du mục -2-. Nhưng đến thứ Sáu lại là một ngày tôi xấu hổ nhất trong đời. Tôi không gặp lại Jasmine (Ms. J) từ khi chúng tôi hoàn tất công việc chiều thứ Sáu. Quả là một tuần "lên voi xuống ngựa" -3-

Tôi trải qua mấy ngày cuối tuần bệ rạc, chắc mẩm thế nào cũng bị chuyển sang bộ phận vệ sinh và tái dụng, dọn rửa lò phản ứng hoặc kỳ cọ mớ rêu mốc. Tôi chẳng màng mặc quần áo hay ăn uống.

Dù gì đi chăng nữa, bây giờ là tối thứ Hai và tôi đang viết để kể cậu nghe sự thể của ngày đầu trong tuần thứ nhì. Nó cũng không quá tệ. Thật ra tôi bắt đầu cảm thấy phấn chấn trở lại.

Sáng hôm nay, tôi thức dậy trong lòng nặng trĩu. Sau khi dùng xong điểm tâm, tôi lết thếch lê đến phòng làm việc, phỏng chừng cô J đang đợi tôi ở bàn máy nhưng thay vào đó là một phụ nữ trung tuần đẫy đà với mớ tóc điểm sương ngang vai và nụ cười hiền từ, nụ cười ít nhiều còn phảng phất dấu ấn của những năm tươi trẻ ngày trước. Lúc nhìn tôi, mắt bà nhấp nháy và bà cười khẽ. "Cậu hẳn là Alphonse," bà nói, đưa tay ra. "Tôi tên là Jean. Tôi nghe khá nhiều về cậu. Cậu đói bụng không? Tôi có một ít bánh mì sandwich ngon lắm trong giỏ nếu cậu muốn, hay cậu thích một quả táo hay một trái chuối."

Quả thật có chiếc giỏ to trên sàn cạnh nơi bà ngồi. Trông chừng như nó không chỉ chứa sandwich và chuối. Bối rối, tôi bắt tay bà và nói, "dạ vâng, thưa bà - ý tôi, không thưa bà - ý tôi - vâng, tôi đúng là Alphonse, và không, cám ơn bà đã mời tôi sandwich và tôi - ườm... ờ.... rất vui khi được gặp bà."

Bà nhìn tôi một cách nghiêm khắc nhưng cái nhếch mép phảng phất nét hiền từ của một người mẹ. "Này, chúng ta không lằn khằn với cái mớ "Bà" ngớ ngẩn đó nghe chưa. Tôi nhất định không phải là "Bà" của ai cả. Cậu gọi tôi là Jean thôi, cậu bé thân mến."

Ừm, cám ơn Jean," tôi trả lời. "Vậy cô J đâu nhỉ?"

"Ai thế cậu bé?"

"Èm... Jasmine đó," Tôi trả lời một cách miễn cưỡng. "Lẽ ra cô ta và tôi làm việc chung với nhau."

"Ôi giời, trước giờ tôi chưa hề nghe ai gọi cô ta là cô J cả. Chắc cô nàng buộc cậu gọi như thế phải không? Tôi không nghĩ có ai gọi cô ta gì khác ngoài cái tên Jasmine. Con bé thật đáng yêu, phải không nhỉ? Mà thôi, cậu bé thân mến, hiện tại ông C muốn cô ta lo công việc khác, cho nên từ rày về sau tôi sẽ làm việc với cậu."

"Bà?" tôi đớ người ra. "Ô," tôi lặp bặp. "Ôi..."

"Nào, cậu ngồi đây đi cậu bé thân mến và để tôi nói cho cậu nghe tôi đang nghĩ gì," Jean nói, trong khi khi vỗ nhẹ trên chiếc ghế cạnh bà.

"Đang nghĩ gì?"

"Tôi xem xét chương trình này trong suốt nửa giờ qua - tôi thích làm việc sớm, cậu biết không -- nhưng chẳng phải sớm hơn thường lệ gì đâu -- tôi biết một cậu con trai đang lớn thì cần ngủ và điểm tâm. Mà thôi, tôi rất hài lòng với chương trình này. Cậu có một chuỗi test rất lý thú và phần mã nguồn rất dễ đọc, cấu trúc lại gọn gàng. Nhưng có một điều làm tôi thắc mắc, cậu bé thân mến."

"Ùm -- thắc mắc?"

"Đúng thế cậu bé! Tôi xem xét khắp nơi trong chương trình này và không hề thấy hàm main. Khi nào cậu sẽ viết phần này vậy hở cậu?"

"Ùm, à, Jerry nói là -" tôi lập bặp.

"Ồ, để tôi đoán thử Jerry nói gì. Jerry là một thằng bé đáng yêu," Jean ngắt ngang, đôi mắt bà nhấp nháy. "Nhưng tôi nghĩ đôi khi cậu ta nên cần thêm manh mối để lý giải vấn đề. Nhưng thôi, tôi không nên có ý kiến không tốt về người khác. Jerry là một lập trình viên giỏi, cậu bé thân mến, cậu đừng để ý đến những điều tôi nói nhá. Nào, hãy viết hàm main. Cậu muốn bắt đầu không? Sáng nay mấy ngón tay tôi hơi bị cứng. Hãy nghe tôi khuyên này," bà cười khúc khích "đừng bị lão hoá."

Đớ người ra từ một chuỗi ngôn từ thẳng tuồn tuột của bà, tôi nhón lấy bàn phím và bắt đầu gõ:

```
public void main(
```

"Ô, nào nào, cậu bé thân mến!" Jean ngưng tôi lại. "Cậu làm việc ở đây được bao lâu rồi nhỉ? cậu phải viết một cái test trước - cậu không thể cắm đầu vào viết hàm main ngay như vậy được! Chúng ta sẽ đi về đâu nếu ai cũng viết mấy cái hàm mà không viết tests trước? Tôi có thể cho cậu biết: (chúng ta sẽ ở) trong một quả dưa chua! -4- Không, cậu bé, xoá nó đi và viết cái test trước."

Bà lấy ra từ trong giỏ một đôi que đan và bắt đầu làm việc với một mảnh y phục nhiều màu có hình dáng na ná như một mảnh khăn choàng -5-, khe khẽ ngâm nga một giai điệu đơn tẻ trong khi tôi xoá mớ chữ vừa viết xong và bắt đầu lại.

Làm cách nào để test hàm main? cách tốt nhất là gọi nó và bảo đảm nó làm những gì nó nên làm. Hàm main giải dịch các thông số trên dòng lệnh, bởi thế, gọi nó chỉ đơn thuần là việc chuyển các thông số cho đúng. Thế nên tôi bắt đầu gõ lại:

```
public void testMain() throws Exception {

  SMCRemoteClient.main(new String[]{"myFile.sm"});
```

Jean ghé mắt nhìn và bảo, "tốt đó, nhưng ở đâu ra cái myFile.sm vậy? Mình không thể để nó nằm ngổn ngang như thế được, phải không nào? Không, mình sẽ tạo nó ở

ngay đây, phải không? và đừng quên xoá nó khi mình đã xong nhá cậu bé thân mến. Không gì tệ hại bằng một mớ hồ sơ cũ nằm chổng chơ, lúc nào tôi cũng bảo thế."

Thế rồi tôi tiếp tục gõ:

```
public void testMain() throws Exception {
  File f = createTestFile("myFile.sm", "the content");
  SMCRemoteClient.main(new String[]{"myFile.sm"});
  f.delete();
  File resultFile = new File("resultFile.java");
  assertTrue(resultFile.exists());
  resultFile.delete();
}
```

Jean hoàn tất thêm một dòng (đan) trên mảnh khăn choàng trong khi tôi gõ mã. Bà ngước lên nhìn khi tôi hoàn tất và nói: "Nào, cậu bé, êm rồi đó, nhưng thật tình tôi nghĩ main có thể không đủ thời gian để hoàn thành trách nhiệm trước khi phân đoạn xoá đó có tác dụng. Nên nhớ cậu bé, cậu có hàng lô các socket threads đang chạy và dễ thấy trong mớ thread này có một thread vẫn đang chạy ngay khi main trả về. Không, cậu bé, khoan hẵn làm gì cả ngay lúc này; chỉ ghi nhớ trong đầu thôi. Bây giờ cậu nên viết main, phải không?"

Tôi tự nhủ bà già này khá sắc sảo và bắt đầu viết phần hàm main.

```
public static void main(String[] args) {
  SMCRemoteClient client = new SMCRemoteClient();
  client.setFilename(args[0]);
  if (client.prepareFile())
    if (client.connect())
```

```
    if (client.compileFile())
      client.close();
    else {  // compileFile
      System.out.println("failed to compile");
    }
  else { // connect
    System.out.println("failed to connect");
  }
 else { // prepareFile
   System.out.println("failed to prep are");
 }
}
```

"Ôi chao ơi, xem có thật sướng mắt không! Tôi nghĩ cách cậu chú thích những đoạn điều kiện cách else rất mẫn đạt. Dẫu vậy, tôi không biết nếu cậu thử đảo ngược ý của các đoạn điều kiện cách và dùng chúng như những phần bảo kê thì có dễ đọc hơn không. Đừng, cậu bé thân mến, hẵng khoan đổi nó. Hãy xem nó có chạy được hay không cái đã. Không đáng để thay đổi quá nhiều thứ cho đến khi mình biết chắc chương trình có làm việc hay không, cậu đồng ý không nào? Đầu tiên làm cho nó chạy trước, rồi mới làm cho nó chỉnh sau."

Thế rồi tôi chạy thử cái test, và nó làm việc ngon lành ngay lần đầu.

"Ồ, đúng là ngoạn mục!" Bà ta nói trong khi ghé mắt nhìn. "Bây giờ chúng ta thử thay đổi phần điều kiện cách if."

Thế rồi tôi đổi hàm cho phần điều kiện cách if được bảo vệ.

```
public static void main(String[] args) {
```

```
 SMCRemoteClient client = new SMCRemoteClient();
 client.setFilename(args[0]);
 if (!client.prepareFile()) {
  System.out.println("failed to prepare");
  return;
 }
 if (!client.connect()) {
  System.out.println("failed to connect");
  return;
 }
 if (!client.compileFile()) {
  System.out.println("failed to compile");
  client.close();
  return;
 }
 client.close();
}
```

Jean đặt mớ đồ đan vào giỏ và xem xét kỹ lưỡng đoạn mã. "Rồi, tôi xem nó được hơn một chút rồi đó, tôi không có ý càm ràm phần lặp ở đoạn close. Và tính chất vi phạm trong lối nhập đơn, xuất đơn, mấy cái này hơi bị vướng víu một chút. Dẫu vậy, nó còn bảnh hơn mớ dòng mã nguồn được đẩy vào (từ lề bên trái) -6-, cậu đồng ý không nào? Tất nhiên chúng ta có thể thay đổi ba hàm đó cho phép chúng throw exceptions -7-, nhưng rồi chúng ta phải catch -7- chúng, và thế cũng khá phiền. Thôi, bây giờ cứ để yên như vậy. Nào, tôi nghĩ đã đến lúc giải lao, phải không cậu bé? Cậu thích mang dùm tôi chiếc giỏ xách này vào phòng ăn không? Tôi luôn có thói nhồi nhét vào giỏ nhiều hơn cần thiết và sau ít lâu cái của khỉ này trở nên nặng trình trịch."

-1- Từ này đã được giải thích một lần trong phần 13 - "Một giải pháp tốt hơn". "Micah" ở đây, trong bài này, có lẽ là một loại đặc quyền hoặc một vinh dự lớn lao. Theo tự điển Merriam-Webster thì Micah là tên của một nhà tiên tri người Do Thái ở thế kỷ thứ 8 sau Công nguyên. "Micah" phát triển từ nguyên thủy MIkhAyAh (tiếng Hebrew).

-2- journeymen đã được chú thích ở bài 14 - Transaction Actions. Journeymen có nghĩa bóng chỉ cho những tay lão luyện trong nghề và thích "lang thang" đi tìm những chân trời mới.

-3- "roller-coaster week": roller-coaster là một loại xe trượt chạy vòng vèo theo cấu trúc dựng sẵn của một giàn giáo. Xe lướt trên roller-coaster với vận tốc nhanh và lên xuống theo cấu trúc dựng sẵn. Ở đây, tác giả hình tượng hoá một tuần làm việc của Alphonse như đi roller-coaster chỉ cho một tuần đi qua rất nhanh và ở trạng thái "khi lên, khi xuống". Tôi tạm dịch là một tuần lên voi, xuống ngựa cho gần với tiếng Việt.

-4- in a pickle, một ngạn ngữ chỉ cho tình trạng hoặc môi trường không hay và không dễ thoát ra. "Pickle" là một loại dưa chua trong dấm ví dụ như trái dưa leo ủ chua (người châu Âu, châu Mỹ và châu Úc rất thích ăn loại này). Ăn thì thích nhưng thử tưởng tượng bị ủ trong một trái dưa chua thì sao?

-5- shawl, một loại khăn choàng đầu và vai dành cho mùa lạnh. Có lẽ từ "khăn sô" đi từ chữ shawl này chăng? Từ đồng nghĩa tiếng Pháp là "châle", cũng đọc nôm na là "sô". Chữ shawl tiếng Anh có nguồn gốc từ chữ "shAl" tiếng Persian (người Iran cổ đại). Tôi chưa tìm ra được chính xác nguồn gốc chữ "khăn sô" của tiếng Việt. Có nguồn cho rằng "khăn sô" đi từ gốc vải sô chuyên dùng làm khăn choàng, cũng có nguồn cho rằng "khăn sô" là một dạng từ vay mượn từ tiếng nước ngoài. Ai có hứng thú (và thời gian) khảo cứu (về ngôn ngữ học), xin đóng góp kết quả và ý kiến.

-6- indentation, lối đẩy dòng chữ thụt vào từ lề trang giấy (từ bên trái hoặc bên phải).

-7- throw exceptions và catch exceptions, có lẽ dân lập trình nhón nhén chút đỉnh đến Java hẳn biết các thuật ngữ này. Tôi để nguyên thuật ngữ này mà không cố gắng dịch vì có thể làm sai lạc khi cố gắng chuyển ngữ.

The Crafsman 18 Slow and Steady

Chậm mà chắc

"Được rồi cậu bé thân mến, tôi nghĩ đã đến lúc chúng ta bắt đầu làm việc với phần server của chương trình này, phải không cậu bé? Đến lúc này phần client làm việc

có vẻ ngon lành như chúng ta thấy được, và một client mà không có server thì trông cô đơn quá. Cậu đồng ý không nào?"

Jean đã an toạ trong chiếc ghế bành của bà. Trong khi nói, bà với tay vào trong giỏ và lôi ra bộ đồ đan. Chiếc sô (hay cái gì đó) đã dài ra một cách đáng kể sau buổi giải lao (khoảng thời gian tôi biết được về con và cháu của Jean).

"È, server?" Tôi chỉ có thể rặn ra hai từ này để trả lời một tràng phát biểu của Jean.

"Đúng vậy, cậu bé thân mến, server. Server sắp sửa nhận hồ sơ mà cậu đã chăm chú gởi, lưu trữ vào đĩa và biên dịch nó với SMC. Nên nhớ cậu bé, chúng ta đang xây dựng một trình dịch từ xa cho SMC, State Machine Compiler. Bây giờ, cậu bé thân mến ơi, chúng ta nên viết test case đầu tiên ra sao?"

Test case gì đây hỡi? tôi gắng vắt óc nghĩ ngợi về chuyện này. Phía server của chương trình lắng nghe trên một socket và nhận các CompileFileTransaction objects. Những object này chứa các gói -1- hồ sơ trong các sub-objects -2- FileCarrier. Server cần viết những hồ sơ được gói này đến một nơi an toàn trên đĩa, biên dịch chúng và gởi kết quả ngược lại client trong một object gọi là CompilerResultsTransaction.

Điều đầu tiên tôi lo ngại là cách biên dịch các hồ sơ này. Bằng cách nào đó, chúng ta cần gọi trình dịch và đòi hỏi nó biên dịch hồ sơ đã lưu.

"Ùm.... mình có thể viết cái test case cho phân đoạn gọi trình dịch không nhỉ?"

Jean mỉm cười theo lối bà ngoại mỉm cười với đứa cháu chập chững bước đi đầu tiên. "Sao lại không, tất nhiên rồi cậu bé thân mến, khởi đầu ở điểm này thú vị đây. Cậu biết cách gọi trình dịch không? Hãy xem, tôi nghĩ chúng ta chỉ dựng đúng dòng lệnh và gọi nó thôi. Nào, cú pháp dòng lệnh ấy sao nhỉ? Đã lâu tôi không mó đến SMC, tôi chẳng nhớ rõ. Nghe đây cậu bé, gõ thử lệnh này: java smc.Smc. Đúng rồi đó cậu, bây giờ xem thử nó báo lỗi gì? Class def not found? Ồ vâng, chúng ta quên cái classpath. Đừng bị lão hoá nha cậu bé, cậu sẽ bắt đầu quên đủ thứ. Gõ cái này: java -cp C:/SMC/smc.jar smc.Smc -3-. Tốt rồi, vậy thông điệp ấy nói gì vậy?"

Thông điệp trên màn hình báo lỗi cách dùng và mô tả mọi thông số cần thiết để dùng SMC.

"È.. đó là một thông điệp chỉ dẫn cách dùng."

"Đúng rồi cậu bé, đúng rồi. Và hình như muốn gọi SMC, chúng ta dùng lệnh sau: java -cp C:/SMC/smc.jar smc.Smc -f myFile.sm. Đồng ý chớ cậu bé? Sao cậu không thử viết một cái test case dùng để tạo dòng lệnh ấy?"

Thế nên tôi tạo một unit test class tên là SMCRemoteServerTest.

```
public class SMCRemoteServerTest extends TestCase {

  public void testBuildCommandLine() throws Exception {

    assertEquals("java -cp C:/SMC/smc.jar -f myFile.sm",

      SMCRemoteServer.buildCommandLine("myFile.sm"));

  }

}
```

Và tiện thể tôi viết luôn SMCRemoteServer.

```
public class SMCRemoteServer {

  public static String buildCommandLine(String file) {

    return null;

  }

}
```

"Tốt lắm, cậu bé thân mến. Cái test bị hỏng như dự tưởng. Cậu học việc khá lắm. Bây giờ làm cho nó đạt là chuyện đơn giản, phải không nào?"

"Èm.... hẳn nhiên rồi."

```
public static String buildCommandLine(String file) {
  return "java -cp C:/SMC/smc.jar smc.Smc -f " + file;
}
```

"Tốt rồi cậu bé, chạy thử tôi xem nào? Tốt. Ôi! Lạ nhỉ, cái test bị hỏng rồi cậu bé thân mến, chuyện gì đã xảy ra nhỉ?"

Thông điệp trên màn hình là: expected:<.....> but was: <...smc.Smc ...>. Tôi cẩn thận xem xét đoạn mã và nhận ra tôi quên phần smc.Smc trong test case. "È... tôi đoán là tôi viết sai đoạn test."

"Đúng rồi, đúng rồi. Vâng, đôi khi chúng ta viết test sai. Bugs -4- có thể xảy ra bất cứ nơi nào. Đó là lý do tại sao viết cả test lẫn mã nguồn là việc nên làm. Bằng cách này, cậu viết mọi thứ hai lần. Tôi có cậu cháu làm kế toán cho tàu vận tải, và cậu ấy luôn luôn lưu các chi tiết chuyển ngân vào hai hồ sơ kế toán riêng biệt. Cậu ấy gọi nó là gì nhỉ? Ồ, vâng, sổ sách đôi. -5- Cậu ấy nói rằng cách này giúp cậu ngăn ngừa và lục tìm các sai sót. Và đây chính là chuyện chúng ta đang làm, phải không cậu bé thân mến. Chúng ta viết tất cả các hàm hai lần. Một lần trong phần test, và một lần trong mã nguồn. Và nó thật sự giúp chúng ta ngăn ngừa và tìm các sai sót, phải không nào?"

Trong khi bà ta tiếp tục với âm thanh của giọng nói đều đều, đơn tẻ, tôi đã sửa xong test case. Jean là một bà già dễ thương và rất giỏi nữa nhưng lỗ nhĩ của tôi lùng bùng với lối nói không ngừng nghỉ của bà.

```
public void testBuildCommandLine() throws Exception {
  assertEquals("java -cp C:/SMC/smc.jar smc.Smc -f myFile.sm",
```

```
    SMCRemoteServer.buildCommandLine("myFile.sm"));
}
```

Thay đổi này làm đoạn test đạt.

"Tuyệt vời, cậu bé, tuyệt vời; nhưng đoạn mã hơi rối, cậu nghĩ thế không? Hãy dọn dẹp nó trước khi tiếp tục. Dọn đống bừa bộn trước khi chúng bắt đầu (trở nên bừa bộn), tôi luôn nói như thế."

Tôi xem đoạn mã và chẳng thấy có gì bừa bộn cả. Thế nên tôi ngoái lại (về hướng Jean ngồi) và nói: "Èm, bà thích khởi đầu từ đâu vậy?"

"Trời phật ơi, sao hỏi vậy cậu bé, nó đơn giản như chiếc mũi trên khuôn mặt cậu vậy thôi! hàm buildCommandLine đó cần chỉnh đốn một chút. Chuỗi string trong dòng trả về đầy tính giả định và nhất định sẽ làm ai đó đọc đoạn mã này sẽ bối rối. Chúng ta không muốn làm cho ai bối rối cả, phải không nào? Tôi muốn nói là chúng ta không muốn làm như thế. Đây cậu, đưa cho tôi cái bàn đánh."

Bà đặt bồ đồ đan xuống (hình như mảnh đan bắt đầu phần cổ tay của cái khăn choàng) và với sự cố gắng rõ rệt để điều khiển bàn đánh, bà gõ chậm rãi nhưng đầy chủ định như thể mỗi cú gõ làm bà đau đớn. Rốt cuộc bà thay đổi đoạn trả lại như sau:

```
return "java -cp " + "C:/SMC/smc.jar" + " " + "smc.Smc" + " -f " + file;
```

Rồi bà chạy thử đoạn test và nó đạt.

"Rồi đó cậu bé thân mến, cậu thấy chưa? Chúng ta cần thay thế đoạn string ấy bằng các giá trị bất biến để giải thích dòng lệnh đã được hình thành bằng cách nào. Cậu muốn làm thế không, hay để tôi?"

Vẻ đau đớn của bà đã đủ trả lời. Tôi vớ lấy bàn đánh và thêm vào các giá trị bất biến tôi nghĩ là bà muốn đưa vào.

```
public class SMCRemoteServer {

  private static final String SMC_CLASSPATH = "C:/SMC/smc.jar";

  private static final String SMC_CLASSNAME = "smc.Smc";


  public static String buildCommandLine(String file) {

    return "java -cp " +

      SMC_CLASSPATH + " " +

      SMC_CLASSNAME +

      " -f " + file;

  }

}
```

"Đúng rồi cậu bé thân mến, đích thị điều tôi muốn. Cậu không nghĩ rằng như thế sẽ giúp những người khác sẽ dễ hiểu hơn sao? -6- Tôi biết chắc nếu tôi là họ, tôi sẽ cảm thấy dễ hiểu hơn. Bây giờ, cậu bé thân mến, sao cậu và tôi không đi đến phòng giải lao để cho đầu óc nghỉ ngơi đôi chút nhỉ?"

Điều gì đó nảy ra trong đầu tôi.

"Jean, chúng ta chưa làm được tí gì cả!"

"Sao cơ, ý cậu thế nào, chúng ta đã hoàn thành khá nhiều rồi."

Tôi chẳng nghĩ đến chuyện tôi đang nói với ai. Tôi vẫn một mực lảm nhảm một cách đần độn. "Chúng ta chỉ mới xong một cái hàm ngu xuẩn mà thôi. Nếu mình giữ cái

đà "con sên" này -7-, ông C sẽ thuyên chuyển chúng ta đến bộ phận canh tác để dọn chuồng thôi!" -8-

"Thật thế sao cậu bé, tại sao cậu lại nghĩ ngớ ngẩn đến như vậy nhỉ? Tôi làm việc với văn phòng này hơn ba mươi năm, cậu bé thân mến, và chưa bao giờ có ai đề cập đến chuyện dọn chuồng."

Bà ta ngưng chốc lát như thể bà đang tổng hợp các ý nghĩ. Rồi bà nhìn tôi với cái nhìn đôn hậu nhưng nghiêm khắc hiện rõ trên khuôn mặt.

"Alphonse thân mến, cách duy nhất để hoàn thành chương trình này nhanh chóng là làm tối đa những gì cậu có thể. Nếu cậu vội vã, hay nếu cậu đi đường tắt, cậu trả phải trả giá xứng đáng trong giai đoạn tìm lỗi. Ông C trả công cho thời gian của cậu, cậu bé, và ông ấy muốn kết quả tốt nhất cậu có thể làm được. Mã nguồn chính là sản phẩm của cậu, cậu bé thân mến, và ông ấy không muốn trả cho thứ sản phẩm kém chất lượng. Ông ta không muốn nghe rằng cậu hoàn tất được ngày làm do vấy vá cho xong chuyện bởi vì ông ta biết cậu sẽ trả một giá rất đắc cho tính vấy vá. Ông ấy chỉ muốn thứ mã nguồn tốt nhất mà cậu có thể viết được. Ông ta muốn mã nguồn sạch, có nghĩa lý và đã được thử nghiệm cẩn thận ở mức tối đa cậu có thể tạo ra. Và rồi, cậu bé thân mến, ông C thừa biết rằng nếu cậu làm như vậy, cậu sẽ hoàn tất nhanh chóng hơn những cậu ngốc non choẹt kia cứ ngỡ chúng có thể xong chuyện nhanh hơn với thói nhếch nhác. Bây giờ, mình hãy đi làm một chén trà nhỉ?"

-1- encapsulated: một thể trạng được gói bên trong một thể trạng, phương tiện nào khác. Ví dụ, một Vector object "encapsulate" các object bên trong và các o bject này có thể khác nhau về tính chất và thể loại. Hay nói ngược lại, các object bên trong một Vector được encapsulated.

-2- sub-objects: các thể trạng (object) ngầm. Ví dụ, một Vector chứa các object và các object này chứa những object khác bên trong.

-3- Trong nguyên bản tác giả cho phép tải smc.jar từ: http://www.objectmentor.com/resource.../smc_java

-4- Bugs: lỗi và vấn đề trục trặc trong lập trình. Từ "bug" này hết sức thông dụng và đặc thù cho nên tôi dùng từ nguyên thuỷ thay vì cố dịch sang một từ tương đương tiếng Việt (như một số từ khác đã được dùng trong suốt series Craftsman này).

-5- dual entry bookkeeping: hồ sơ kế toán được ghi nhận và lưu giữ hai nơi khác nhau. Trên thực tế, tôi không rõ có phương pháp kế toán như thế này không nhưng đây là chi tiết dùng để liên hệ đến vấn đề viết test và viết code nên được ghi nhận là một chi tiết quan trọng.

-6- Câu nói ở dạng negative (negative form) đặc biệt được nhân vật Jean dùng thường xuyên. Có lẽ tác giả muốn tạo nhân vật Jean này với cá tính rất mềm mỏng và nhẹ nhàng trong khi trao đổi. Tuy nhiên, những điều Jean đưa ra rất xác thực và cần thiết cho dù lối nói của bà ta rất dông dài.

-7- snail's pace: mức độ, tốc độ của con sên. Ý nói sự việc tiến triển rất chậm chạp.

-8- Clean out the Dribin cages: tôi không tìm được nguồn gốc của cụm từ này. Độc giả có ai rõ, xin góp ý.

The Crafsman 19. Tolerance

Kiên nhẫn

Jean đưa tôi đến thang máy để xuống tầng ngầm 36 thuộc cánh gamma, phòng khách của các tay du mục. Bà nói tầng ngầm này -1- làm các khớp xương của bà hết sức dễ chịu. Khi chúng tôi đến nơi, tôi thấy có Jerry, Jasmine và một vài tay du mục đang tụ lại một bàn. Jean cũng thấy họ và rảo bước đến nơi họ ngồi. Tôi miễn cưỡng đi theo.

"Chào các cô, các cậu thân mến!" Bà thốt lên mừng rỡ như thể không gặp họ sau nhiều tuần lễ. Bọn họ đều chào lại cũng với vẻ thân mật (như bà dành cho họ). Thế rồi bà cáo lỗi và rảo bước về chiếc ghế đấm bóp bỏ trống, để lại một mình tôi đối diện với hai cựu huấn luyện viên.

"Chào Jerry; chào cô J. Đằng ấy có khoẻ không?"

Jerry trông có vẻ sửng sốt. "Cái của khỉ 'cô J'. gì đây nhỉ?"

Jasmine vội vàng lảng đi thật nhanh cho xong chuyện: "Quên cái chuyện 'Ms. J' đi cao thủ. Chuyện này đã quá đủ." Đôi mắt nàng chưa bao giờ dịu như vậy, và tôi nhận ra chính mình cũng không thể trả lời nàng một cách khéo léo và gọn gàng.

Chiếc xe phục vụ cà-phê đi ngang. Tôi mừng rỡ vớ ngay một cốc vì có dịp may để đánh trống lảng và rồi tôi lại tiếp tục đối diện với các tay cựu huấn luyện viên này.

"Jean đối xử với cậu thế nào hở cao thủ?" Jasmine hỏi.

Jerry chêm vào: " Ừa, làm sao mày "gỡ" được một "món" của bà ta vậy?"

Sau khi ở trong hoàn cảnh không thể nói hơn ba chữ một lần suốt cả buổi sáng với Jean, nỗi bực dọc trong lòng tôi vỡ tung ra.

"Tôi không "gỡ" gì hết, bà ấy bất thình lình xuất hiện sáng nay. Thật tình mà nói, làm việc với bà tôi thấy hơi khó chịu. Bà ta lúc nào cũng gọi tôi là 'cậu bé thân mến'; và bà nói quá nhiều, nghĩ giải lao cũng quá nhiều. Chúng tôi không hoàn tất được bao nhiêu việc cả. Tôi không chắc tôi có muốn tiếp tục làm việc với bà ta hay không."

Jasmine và Jerry nhìn tôi, họ nhìn nhau, rồi đột nhiên phá ra cười. Jerry trở lại bình thường một cách nhanh chóng nhưng Jasmine không thể tự chủ được. Mỗi khi nhìn tôi, nàng lại cười phá lên; tiếng cười hao hao như giọng con hải sư gọi bạn -2-.

"Gì vậy?" tôi hỏi.

Jerry dẫn tôi qua một bên trong khi Jasmine tiếp tục tuôn ra hàng tràng cười lảnh lót. "Alphonse, mày không biết mày may mắn như thế nào. Bọn tao ở đây đứa nào cũng sẵn sàng đánh đổi bất cứ điều gì -3- để được làm việc với Jean. Tao biết bà ấy hơi khác thường nhưng đừng nên lầm với cái vẻ "bà ngoại" bên ngoài của bà ta. Bà ấy là một trong những tay thiện nghệ nhất và mày sẽ học được rất nhiều từ bà."

Tôi ngớ ra nhưng không thể đối đáp thêm được tí gì vì ngay khi ấy Jean khập khiễng từ chiếc ghế đấm bóp đi đến.

"Ơn trời, tôi cảm thấy khoẻ hơn rất nhiều."

Trao đổi với Jerry tôi tìm được những thông tin nhiều hơn cần thiết cho nên tôi đổi chủ đề bằng cách đề nghị gọi người phục vụ cà phê.

"Ồ thôi cậu bé. Tôi nghĩ chúng ta nên về lại tầng 6 thiếu tiện nghi của chúng ta và tiếp tục làm việc với SMCRemote, cậu có nghĩ thế không? Chúng mình còn quá nhiều thứ phải lo trong ngày hôm nay và hẳn nhiên chúng ta sẽ chẳng thực hiện được nếu chúng ta ở đây, phải không nào? .... Không hiểu sao Jasmine lại phát ra những âm thanh khủng khiếp vậy nhỉ? Âm thanh giống như con thú nào đó đang giẫy chết vậy. Jasmine, uống chút nước đi cô bé thân mến..."

Chúng tôi dùng thang máy đi về phòng làm việc. Tầng lầu cao này làm Jean chậm chạp hẳn. Ở dưới phòng khách bà ta nhanh nhẹn hơn rất nhiều.

Sau khi yên ổn ngồi vào bàn máy, bà nói: "Nào, Alphonse thân mến, tôi nghĩ chúng ta nên xem thử có thể chạy được đoạn lệnh mình đã tạo ra không. Cậu nghĩ sao?"

Tôi đã ngẫm nghĩ cách thực hiện nên tôi đồng ý ngay. "È... vâng."

"Tốt lắm cậu bé thân mến. Bây giờ thế này, đoạn lệnh mình sắp chạy sẽ gọi chương trình biên dịch SMC, tôi nghĩ việc đầu tiên mình cần làm là tạo ra một đoạn mã đơn giản để trình dịch đó đọc. Thế, hãy viết một cái test dùng để tạo hồ sơ này đi, rồi gọi trình dịch và kiểm tra xem trình dịch có tạo ra hồ sơ xuất đúng hay không."

Bà lại lôi ra bộ đồ đan len và không hề tỏ ý muốn động đến bàn đánh, bởi vậy tôi vớ lấy nó và bắt đầu gõ:

```
public void testExecuteCommand() throws Exception {

  File sourceFile = new File("simpleSourceFile.sm");

  PrintWriter pw = new PrintWriter(new FileWriter(sourceFile));

  pw.println("

}
```

Không biết phải tiếp tục thế nào nên tôi nhìn bà với ý chờ đợi.

"Tốt lắm cậu bé thân mến. Đây, để tôi gõ vào cú pháp SMC cho cậu."

```
public void testExecuteCommand() throws Exception {

  File sourceFile = new File("simpleSourceFile.sm");

  PrintWriter pw = new PrintWriter(new FileWriter(sourceFile));

  pw.println("Context C");

  pw.println("FSMName F");

  pw.println("Initial I");

  pw.println("{I{E I A}}");

  pw.close();

}
```

"Nó có nghĩa thế nào vậy Jean?"

"À, cậu bé thân mến, với mục đích riêng của chúng ta thì nó có nghĩa là trình dịch sẽ tạo ra hồ sơ tên là F.java. Lúc này cậu chỉ cần biết ngần ấy thôi. Tuy nhiên, khi cậu trở về phòng riêng của mình, cậu nên tìm kiếm tài liệu SMC và tham khảo thêm; việc này là việc cần thiết. Tôi viết tài liệu này nhiều năm về trước đó cậu bé thân mến và tôi vẫn nghĩ nó là một trong những tài liệu hay của tôi. Nó gọi là "Care and Feeding of the State Map Compiler" -4-. Bây giờ mình xem thử có chạy được lệnh biên dịch ấy không."

Tôi không chắc phải làm gì để chạy lệnh này; nhưng tôi học được từ Jerry và Jasmine thông thường cách hay nhất là chỉ viết các cú gọi diễn tả ý định của mình. Bởi thế tôi tiếp tục gõ:

```
public void testExecuteCommand() throws Exception {

  File sourceFile = new File("simpleSourceFile.sm");

  PrintWriter pw = new PrintWriter(new FileWriter(sourceFile));

  pw.println("Context C");

  pw.println("FSMName F");

  pw.println("Initial I");

  pw.println("{I{E I A}}");

  pw.close();

  String command = SMCRemoteServer.buildCommandLine("simpleSourceFile.sm");

  assertEquals(true, SMCRemoteServer.executeCommand(command));


  File outputFile = new File("F.java");

  assertTrue(outputFile.exists());

  assertTrue(outputFile.delete());

  assertTrue(sourceFile.delete());

}
```

"Mỹ mãn! Alphonse. Tôi nghĩ rằng đoạn trên nắm bắt phần test một cách rất đáng khen. Chúng ta chưa viết executeCommand nhưng chắc chắn mình có thể diễn đạt cách mình muốn nó được gọi ra sao, phải không nào. Bây giờ chúng ta hãy tạo stub -5- cho bước này và xem đoạn test bị hỏng. Lúc nào tôi cũng thấy thú vị khi thấy chúng hỏng, cậu có nghĩ thế không?"

Thế rồi tôi bấm vào executeCommand và chọn "Create Method", và IDE của tôi -6- tạo stub method tôi muốn. Và rồi, y như thật, phần test bị hỏng.

```
public class SMCRemoteServer {

 ...

 public static boolean executeCommand(String command) {

  return false;

 }

}
```

"Được rồi cậu bé thân mến, hãy làm cho cái test ấy đạt nhé."

Jean lại có vẻ mệt mỏi. Tôi biết bà sắp muốn nghỉ tay thêm lần nữa. Đã gần đến giờ ăn trưa, nên tôi hy vọng bà có thể nán lại cho đến lúc ấy. Tôi nhanh chóng rảo xuyên qua javadocs để tìm cách thực thi một lệnh. Jean thấy vậy bèn nói:

"Trong Runtime class đó, cậu bé. Thế rồi tôi gõ đoạn mã tôi nghĩ nó chạy.

```
public static boolean executeCommand(String command) {

 Runtime rt = Runtime.getRuntime();

 try {

  rt.exec(command);
```

```
   return true;
  }
  catch (IOException e) {
   return false;
  }
}
```

Nhưng khi tôi chạy phần test, nó không tìm được hồ sơ xuất. Tôi tìm trong thư mục và quả thật, F.java không có ở đó.

"Sao nó hỏng vậy, Jean?"

Bà nhìn lên và bảo: "Cậu không đợi cho process chấm dứt đó cậu bé thân mến. Khi cậu thực hiện một lệnh, nó tạo ra một process mới chạy đồng thời với process của cậu. Cậu phải đợi cho nó hoàn tất trước khi thoát ra khỏi executeCommand."

Tôi tham khảo phần Runtime.exe trong JavaDoc và thấy nó trả lại một object Process. Tôi cũng thấy rằng mình có thể đợi object Process hoàn tất và có thể truy khảo tình trạng thoát -7- của nó. Bởi thế tôi tạo những thay đổi như sau:

```
public static boolean executeCommand(String command) {
  Runtime rt = Runtime.getRuntime();
  try {
    Process p = rt.exec(command);
    p.waitFor();
    return p.exitValue() == 0;
  }
  catch (Exception e) {
```

```
    return false;
  }
}
```

Lần này phần test đạt.

"Cũng không khó lắm." Tôi phát biểu.

"Tất nhiên là không rồi cậu bé thân mến. Chúng ta chỉ chạy một cái lệnh thôi mà. Tất nhiên nó sẽ trở nên phức tạp hơn một chút khi chúng ta phải nắm bắt thông điệp mà trình dịch thường in ra trên console -8-. Chúng ta sẽ phải gắn vào standard output và standard error -9- của Process. Nhưng hãy làm chuyện ấy sau giờ ăn trưa cậu bé thân mến, tôi bắt đầu cảm thấy đói, cậu có thấy vậy không?"

Tôi thở dài và đứng dậy. Tôi cảm thấy dường như chúng tôi vẫn tiến triển chậm chạp. Tuy vậy, xét lại thì tôi thấy chúng tôi đã hoàn tất phần client và làm cho server chạy được trong nửa ngày đầu làm việc với Jean. Chúng tôi không dành tí thời gian nào cho việc tìm lỗi. Có lẽ chúng tôi tiến triển nhanh hơn tôi nghĩ.

Jean bỏ bộ đồ đan vào giỏ và giăng ra chiếc áo len. "Đây, cậu bé thân mến, thử cái này vào xem."

Nếu không còn gì đáng để nói thì chiếc "đồ" này của Jean dạy cho tôi thêm tính kiên nhẫn.

-1- Nguyên văn "low-g", một dạng tiếng tiếng lóng chỉ cho "lower ground". Đây là lối nói rất Mỹ.

-2- "sounded like a sea lion calling to its mate" tạm dịch là "tiếng cười hao hao như giọng con hải sư gọi bạn". Có lẽ tác giả muốn phần nào bộc tả tình cảm của Alphonse lúc ấy bằng lối so sánh đầy hình tượng này. Theo tôi, giọng con hải sư gọi bạn quả thật khó có thể "cảm" nổi (nhưng có lẽ lại hấp dẫn đối với giống hải sư chăng).

-3- "Give our eyeteeth", một thành ngữ chỉ cho sự đánh đổi rất đắt giá.

-4- "Care and Feeding of the State Map Compiler", tạm dịch là "Chăm sóc và bồi dưỡng trình dịch State Map". Trong nguyên bản, tác giả cho đường dẫn đến: http://www.objectmentor.com/resources/downloads/bin/smcJava.zip để tải smcJava.zip.

-5- stub theo nguyên bản, tạm dịch là nội đệm và có lẽ skeleton sẽ là ngoại đệm (dịch theo ngữ cảnh). Đối với những ai tiếp xúc với RMI (Remote Method Invocation - đọc thêm ở http://java.sun.com/products/jdk/rmi/) và J2EE chắc không lạ với khái niệm "stub" và "skeleton".

Tổng quát mà nói, "stub" là một class nội bộ (local class) có cùng interface với một class tầm xa (remote class). Với RMI, class tầm xa là class được gọi để thực hiện một công tác nào đó. Để thực hiện một công tác này, bạn chỉ cần gọi "stub" nội bộ mà không cần phải quan tâm đến chuyện công tác này được thực hiện bởi một class nào đó từ xa.

Trong khi đó, "skeleton" là một class tầm xa dùng để tiếp nhận thông điệp và lo liệu cú gọi đến một class nào đó trên một máy tầm xa và class này chính là class thực hiện công tác. Chỉ cần ghi nhận tổng quát: stub dành cho local và skeleton dành cho tầm xa trong cơ chế RMI.

-6- IDE, viết tắt từ Integrated Development Environment (không phải là Intergrated Drive Electronics dành để chỉ cho ổ cứng), tạm dịch là bộ tích hợp môi trường phát triển. IDE là một bộ công cụ dùng để phát triển chương trình. Ví dụ, Java Netbeans, Eclipse, VC++ là các IDE. Chỉ cần nhớ IDE là đủ :)

-7- exit status, tạm dịch là "tình trạng thoát". Exit status là thuật ngữ quen thuộc với những ai đã từng lập trình, nó dựa trên các mã số đã được quy định trước để xác định một chương trình sau khi thoát ra thuộc tình trạng nào.

-8- console, một thuật thông dụng trong ngành điện toán. Trước đây "console" là một màn hình gắn liền với bàn đánh trong môi trường mainframe hoặc các hệ thống UNIX (không dùng giao diện đồ hình). Mỗi màn hình là một "console". Cho đến ngày nay, các hệ điều hành hiện đại kèm theo giao diện đồ hình nhưng vẫn còn phương tiện để mở lên một "console" (như DOS prompt trên Windows hoặc term trên *nix nói chung). Nói theo phương diện kỹ thuật, console là một giao diện, hay một cơ chế đứng giữa chương trình làm việc và hệ điều hành.

-9- "standard output", "standard error" và "standard in", đôi khi còn viết tắt là "stdin", "stdout", "stderr". Các thuật ngữ này được dùng để chỉ cho cơ chế xử dụng các thiết bị xuất, nhập dữ liệu hoặc hiển thị lỗi trên console (ở trên). Với stdin, stdout và stderr trong môi trường Java, nên tham khảo tài liệu "The ins and outs of standard input/output" của Jeff Friesen ở http://www.javaworld.com/javaworld/jw-03-2001/jw-0302-java101.html

The Crafsman 20. SMCRemote Part X Backslide.

Chổng gọng -1-

Tôi trở lại phòng làm việc sau buổi trưa nhưng Jean không có đó. Chẳng có một mẩu tin hay e-mail của bà để lại; và ngay cả giỏ đồ đan của bà cũng chẳng thấy tăm hơi. Sau vài phút, tôi quyết định ngồi xuống và tiếp tục làm việc với SCMRemoteServer.

Cho đến lúc này server chẳng phục vụ gì hết. Nó chỉ có vài cái hàm dùng để dựng và thi hành dòng lệnh SMC. Tôi nghĩ đúng ra mã nguồn của server phải mở một socket và tiếp nhận các đường nối từ SMCRemoteClient mà chúng tôi đã viết lúc trước.

Tôi khá ngán ngẩm với đà làm việc của chúng tôi hôm nay. Mất cả buổi sáng mà chúng tôi chỉ hoàn thành hai cái hàm bé tẹo thêm vào mấy cái unit test. Tôi muốn thấy sự tiến triển. Bởi thế tôi vớ lấy bàn đánh và bắt đầu gõ.

"Đầu tiên," tôi nghĩ, "các server này cần phục vụ gì đó. Thế thì hãy dùng SocketServer class mà Jerry và tôi xong tuần trước."

Làm cho server phục vụ khá đơn giản. Tôi chỉ cần viết một constructor dùng để tạo một SocketService object và chuyển vào trong một SocketServer. Rồi SocketServer.serve hẳn sẽ tự động được gọi khi SMCRemoteClient muốn truy cập.

```
public SMCRemoteServer() throws Exception {

  service = new SocketService(9000, new SocketServer(){
```

```
    public void serve(Socket socket) {
      // SMCRemoteClient has connected.
    }
  });
}
```

Tôi xem trong mã nguồn của SMCRemoteClient và thấy rằng client đợi một string được gởi đi (từ server) sau khi đã kết nối. string đó bắt đầu bằng "SMCR". Thế nên tôi viết đoạn string đó trong method serve().

```
public void serve(Socket socket) {
  // SMCRemoteClient has connected.
  try {
    ObjectOutputStream os =
    new ObjectOutputStream(socket.getOutputStream());
    os.writeObject("SMCR");
  }
  catch (IOException e) {
  }
}
```

Kế tiếp server cần đọc một CompileFileTransaction từ client. Nó cần viết các hồ sơ chứa trong transaction ấy, gọi trình dịch và sau đó trả về kết quả (các hồ sơ) bên trong CompileResultsTransaction. Viết nên nó có vẻ không khó mấy, thế....

```
public void serve(Socket socket) {
  // SMCRemoteClient has connected.
  try {
    ObjectOutputStream os =
      new ObjectOutputStream(socket.getOutputStream());
    os.writeObject("SMCR");
    ObjectInputStream is =
      new ObjectInputStream(socket.getInputStream());
    CompileFileTransaction cft =
      (CompileFileTransaction)is.readObject();
    String filename = cft.getFilename();
    cft.sourceFile.write();
    String command = buildCommandLine(filename);
    executeCommand(command);
    //OK, what file do I put into the Result?
  }
  catch (Exception e) {
  }
}
```

Hừm. Biên dịch hồ sơ có vẻ không khó lắm nhưng tôi nên đưa vào kết quả transaction của một hồ sơ xuất ra sao? Tên nó là gì? Tôi nhớ hồi sáng nay Jean đã hướng dẫn tôi viết một cái test để gọi trình dịch. Tôi xem lại phần test ấy và thấy hồ sơ nhập có cái đuôi là a.sm, và hồ sơ xuất có cái đuôi a.java . Thế nên tôi chỉ thay thế ".sm" bằng ".java" cho tên hồ sơ.

```
public void serve(Socket socket) {
  // SMCRemoteClient has connected.
  try {
    ObjectOutputStream os =
      new ObjectOutputStream(socket.getOutputStream());
    os.writeObject("SMCR");
    ObjectInputStream is =
      new ObjectInputStream(socket.getInputStream());
    CompileFileTransaction cft =
      (CompileFileTransaction)is.readObject();
    String filename = cft.getFilename();
    cft.sourceFile.write();
    String command = buildCommandLine(filename);
    executeCommand(command);
    //Figure out the file name.
    String compiledFile = filename.replaceAll("\.sm", ".java");
    CompilerResultsTransaction crt =
      new CompilerResultsTransaction(compiledFile);
    os.writeObject(crt);
    socket.close();
  }
  catch (Exception e) {
  }
}
```

Nhìn có vẻ ngon lành. Bây giờ tôi chỉ cần khởi động server và chạy client. Đơn giản thôi. Thế rồi tôi tạo hồ sơ nguồn trong thư mục của tôi có cái tên là F.sm, y như cái Jean muốn tôi tạo ra sáng nay.

```
Context C

FSMName F

Initial I

{I{E I A}}
```

Và rồi tôi viết một hàm main trong SMCRemoteServer

```
public static void main(String[] args) throws Exception {

  SMCRemoteServer server = new SMCRemoteServer();

}
```

Sau đó tôi chạy thử. Và nó chỉ treo ngay đó, đợi cho client truy cập. *Cái này* mới thật là thích! Ít ra tôi đã *hoàn thành* được cái gì đó! Thế rồi tiếp theo đó tôi chạy client với "F.sm" làm thông số. Và nó *CHẠY*! Hay nói cách khác, nó thoát ra sau nhiều giây đồng hồ với tín hiệu thoát -2- bình thường và chẳng có bất cứ thông báo lỗi nào cả từ client cũng như server. Quá thích!

Nhưng nó đã làm những gì? tôi không thể xác định ngay được. Bởi thế, tôi thử kiểm tra thư mục và thấy một hồ sơ F.java nằm chễm chệ ở đó! Nó có đúng ngày trên đó, thế có nghĩa nó đã được tạo ra từ lần client chạy vừa rồi. Quá tuyệt! Tôi mở hồ sơ ra xem và nó trông giống như mã Java đã được tạo nên. Nó còn nói là được SMC tạo ra. Mã nguồn của tôi chạy được! -3-

Đã mấy tuần nay tôi chưa hề cảm thấy vui sướng thế này, thật sự mà nói, kể cả từ trước khi làm việc với Jerry. Đây mới đúng là lập trình! Tôi dâng tràn cảm giác muốn chóng làm cho mã nguồn chạy. Tôi là "độc cô cầu bại"! -4- Tôi đứng lên và nhún

nhảy một vòng quanh chiếc ghế, miệng lảm nhảm "Ôi! ôi! tôi là một lập trình viên. Ôi!"

Jerry hẳn ở đâu đó gần bên bởi vì gã bước vào phòng làm việc ngay lúc ấy. "Ê Alphonse, mày đang rộn ràng chuyện gì vậy?"

"Ồ, chào Jerry! Xem này! tôi mới làm cho SMCRemoteServer chạy được!"

"Thật vậy à? Vậy là chiến lắm đó." Vẻ nhìn trên mặt gã trông buồn cười -- như thể gã không tin tôi. Thế nên tôi chỉ cho gã xem. Tôi chỉ cho gã xem server ấy làm việc ra sao. Tôi chạy client một lần nữa. Tôi chỉ cho gã xem hồ sơ F.java với đúng ngày giờ được tạo ra. Thậm chí tôi chỉ cho gã xem hồ sơ này chứa mã java.

Jerry nhìn tôi gần như kinh hãi, và rồi gã liếc ra cửa một cách sợ sệt. "Alphonse, mấy cái tests của mày đâu?"

"Jerry, ông không cần mấy cái unit tests cho những thứ đơn giản như thế này. Nhìn xem, chỉ có một chục dòng mã nguồn hoặc hơn thôi mà. Jerry, tôi đã có tiến bộ đây này. Tôi đã *hoàn tất* -5- dăm ba điều. Và nó không cần phải mất cả ngày để hoàn thành! Tôi nghĩ mấy ông bà phí quá nhiều thời gian với mấy cái unit test!"

Jerry nhìn tôi chằm chặp vài giây như thể gã không thể tiếp thu những điều tôi bảo gã. Thế rồi gã đóng cửa phòng làm việc lại và ngồi xuống bên cạnh tôi.

"Alphonse, bà Jean thấy mấy cái này chưa vậy?"

"Chưa, tôi chưa thấy bà trở lại sau giờ trưa."

"Xoá nó đi, Alphonse."

"Xoá cái gì?"

"Xoá đoạn mã mày vừa viết xong đó. Nó vô dụng."

"Tôi đã từng đụng kiểu phản ứng này trước đây. Tôi cong cớn bảo: "Ôi, thôi đi Jerry! nó chạy mà! làm sao nó có thể vô dụng?"

"Mày có chắc nó chạy không?"

"Thì chính ông cũng thấy đó!"

"MÀY có chắc nó chạy không?"

"Tất nhiên là nó chạy. Hồ sơ F.java chính là bằng chứng!"

"Alphonse, tại sao ngày giờ của hồ sơ F.sm giống y hệt ngày giờ trên hồ sơ F.java vậy?"

"Cái gì?" Tôi nhìn vào thư mục thì quả thật, chúng giống nhau cho đến từng giây. Nhưng không thể hiểu nổi. Tôi viết hồ sơ F.sm bằng tay nhiều phút trước đây. Tại sao nó có cùng ngày giờ với hồ sơ F.java vừa được tạo ra mấy giây trước, khi tôi biểu diễn cho Jerry xem? Tôi chăm chăm nhìn nó một đỗi và thú nhận tôi không biết tại sao.

"Xoá mã nguồn này đi Alphonse. Mày không hiểu gì hết."

"Ôi, thôi đi Jerry, tôi hiểu mà. Tôi chỉ không tìm ra lý do tại sao mớ ngày giờ lại không đúng thôi. Nó chạy mà Jerry... nó chạy." Nhưng tôi không còn dám chắc nữa. *Tại sao phần ngày giờ ấy phải khác nhau nhỉ*?

"Alphonse, có thể nào mày chạy client và server từ cùng một thư mục không?"

"Ui da..." Tôi không ấn định thư mục khác nhau cho chúng nên tôi đoán chúng chắc đã chạy trong cùng một nơi. ".... Tôi đoán là thế."

"Vậy cả client và server đều đọc và viết cùng hồ sơ trong cùng một thư mục?"

"... ôi.."

Jerry gật đầu với vẻ đắc thắng. "Ừa."

Tôi gật đầu. "OK, Jerry, ông có lý lắm. Tôi không hiểu hết những gì xảy ra. Nhưng hãy xem, hồ sơ F.java nằm chường ra kìa. Nhất định phải có *cái gì đó* chạy được!"

"Thế thì sao? Mày không biết cái gì chạy và cái gì không. Mày không hiểu mày đã làm gì. Mấy thứ này trông có vẻ như chạy được theo kiểu chó ngáp nhầm ruồi. -6- Có thể nào, ví dụ như hồ sơ F.java bị sót lại từ cái unit test trước không?"

Có thể lắm! Chu choa, Jean và tôi làm cho hệ thống viết một cái F.java sáng nay! "Quỷ tha! Vâng, có thể lắm -- nhưng không e không chắc là như vậy!"

"Xoá đoạn mã đi Alphonse. Toàn rác rưởi."

Tôi nhìn gã chằm chằm. Tôi đã tiến triển quỷ tha nó đi! Bây giờ gã lại muốn tôi xoá hết những dấu ấn *tiến triển* ấy. Nhưng gã ấy đúng. Tôi không hiểu đoạn mã. Và tôi chẳng có tí test nào có thể chứng minh từng bước một là mọi thứ chạy đúng nhưng dự tưởng. Nếu mã nguồn của tôi thực sự làm việc đúng (và bây giờ tôi bắt đầu thật sự băng khoăn không biết nó làm việc đúng được mấy phần) nó chạy được do may mắn hơn do thiết kế đàng hoàng.

"Xoá đoạn mã đi Alphonse. Mày chớ có để bà ấy thấy mớ mã nguồn đó. Ngay lúc này, bà ta hết lòng quan tâm -7- đến mày, và mớ mã này sẽ làm bà thất vọng đó."

Lập trường của tôi rốt cuộc hỏng bét. Đôi vai trễ xuống, cái đầu cứng đơ và tôi chồm sang, xoá hết đoạn mã. Jerry bước ra khỏi phòng, đầu ngúc ngoắc.

Vài phút sau, bà Jean bước vào. "Chào Alphonse thân mến. Tôi xin lỗi vì chậm trễ vì tôi bị dính vào câu chuyện với người bạn cũ và bọn tôi sa đà vào chuyện so hình của mấy đứa cháu. Tôi mê khoe hình mấy đứa cháu tôi lắm. Cậu thấy chúng chưa nhỉ, cậu bé thân mến? Ồ, đừng để ý đến tôi, chúng ta có việc cần phải làm. Cậu làm gì trong khi tôi bị "tạm giam" vậy?"

"Không làm gì cả Jean, tôi chỉ đợi bà thôi."

-1- Backslide: chỉ cho tình trạng hụt hẫng, bị rớt xuống một mức thấp hơn. Tạm dịch là "chổng gọng" cho thêm phần... dí dỏm.

-2- Exit Code: tạm dịch là tín hiệu thoát. Exit code là thuật ngữ quen thuộc với những ai đã từng lập trình, nó dựa trên các mã số đã được quy định trước để xác định một chương trình sau khi thoát ra thuộc tình trạng nào.

-3- Đoạn này tác giả dùng rất nhiều dấu chấm thang (!) sau mỗi câu Alphonse thốt ra để nhấn mạnh tình trạng cảm xúc của Alphonse lúc này. Thông thường dấu chấm thang được xếp loại và diện "kỵ dùng" quá nhiều trong văn viết nhưng trong phần này, dấu chấm thang được huy động tối đa và có tác dụng rất thích đáng.

-4- Invincible: không thể bại. Tạm dịch là "độc cô cầu bại" cho gần với tinh thần "kiếm hiệp" của dân Á châu nói chung.

-5- Chú ý các chữ hoặc cụm chữ in nghiêng trong bài này. Chúng dùng để nhấn mạnh cũng như để tạo kịch tính trong câu chuyện.

-6- working by accident: sát nghĩa là "chạy được nhờ may rủi". Tạm dịch là "chạy được theo kiểu chó ngáp nhầm ruồi" để cố gắng lột tả cá tính của Jerry.

-7- think the world of someone: hết lòng hết dạ với ai đó, để hết tâm ý đến ai đó.