**Thợ lành nghề #11: Dùng hàm main để làm gì? (SMCRemote – phần 1)**

Bạn có thể tải mã nguồn của bài viết trước ở [đây](http://www.objectmentor.com/resources/craftman/SDSocketServerR5_DanglingThreads.zip).

*Ngày 18 tháng 2 năm 2003*

Trong đầu tôi cứ cân nhắc mãi mớ threads treo lủng lẳng trong khi ăn món mì ống spaghetti 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, gửi mã nguồn đến server và gửi ngược lại tệp tin đã biên dịch.”

Jerry nhìn tôi chờ đợi và hỏi, “mày nghĩ mình bắt đầu 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. “Bắt đầu từ cái nhìn của người dùng luôn luôn là một điều hay. Thế thì đâu là cách đơn giản nhất người dùng có thể sử dụng công cụ này?”

“Anh ta có thể yêu cầu một tệp tin nào đó được biên dịch. Lệnh ấy có thể như thế này.” tôi viết lên tường như sau:

java SMCRemoteClient myFile.sm

“Coi được đó,” Jerry nói. “Vậy mình bắt đầu sao đây?”

Mì spaghetti đã ấm trong dạ dày, và 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];

}

}

“Xin lỗi!” Jerry ngắt ngang. “Mày có kiểm thử cho nó không?”

“Ý ông là sao?” tôi hỏi một cách thiếu kiên nhẫn. “Mã nguồn này thuộc dạng lẻ tẻ – sao mình phải viết kiểm thử cho nó làm chi?”

“Nếu mày không viết một kiểm thử 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 cuộc tôi nói.

“Vậy sao?” Jerry trả lời. “Tao không thấy thuyết phục cho lắm. Hãy thử một lối khác xem sao.”

Jerry 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 tệp tin 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.

Ðầu tiên gã viết một đoạn mã của khung làm việc kiểm thử mà giờ đã quen thuộc:

import junit.framework.\*;

public class TestSMCRemoteClient extends TestCase {

public TestSMCRemoteClient(String name) {

super(name);

}

}

Gã biên dịch và chạy, nắm chắc rằng kiểm thử bị hỏng vì thiếu kiểm thử, và rồi gã thêm kiểm thử sau:

public void testParseCommandLine() throws Exception {

SMCRemoteClient c = new SMCRemoteClient();

c.parseCommandLine(new String[] { &quot;filename&quot; });

assertEquals(&quot;filename&quot;, c.filename());

}

“Được thôi,” tôi nói. “Có vẻ như ông lấy đối số của dòng lệnh bằng hàm parseCommandLine thay vì dùng main, nhưng phiền như thế làm gì?”

“Làm thế để tao có thể kiểm thử,” Jerry trả lời.

“Nhưng chẳng có gì để mà kiểm thử,” tôi cằn nhằn.

“Ðiều đó có nghĩa là viết kiểm thử thực sự rẻ.” 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 kiểm thử 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 nói “Tốt, nó thành công rồi”

Sau đó hắn lặng lẽ viết trường hợp kiểm thử tiếp theo.

public void testParseInvalidCommandLine() {

SMCRemoteClient c = new SMCRemoteClient();

boolean result = c.parseCommandLine(new String[0]);

assertTrue(&quot;result should be false&quot;, !result);

}

Tôi lẽ ra phải biết hắn sẽ làm như thế. Hắn đã chỉ cho tôi thấy rằng việc viết kiểm thử mà tôi cho rằng không cần thiết là một ý 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 kiểm thử cho riêng nó.” Thế rồi tôi vớ lấy bàn phím và làm cho kiểm thử đạt.

public boolean parseCommandLine(String[] args) {

try {

itsFilename = args[0];

} catch (ArrayIndexOutOfBoundsException e) {

return false;

}

return true;

}

Cân nhắc kỹ lưỡng, tôi tái cấu trúc biến c và khởi tạo cho nó trong hàm setUp. Các kiểm thử đều đạt.

Trước khi Jerry có thể đề nghị trường hợp kiểm thử tiếp theo, tôi nói, “Rất có khả năng tệp tin không tồn tại. Chúng ta nên viết một kiểm thử chứng minh mình có thể xử lý trường hợp ấy được.”

“Quả vậy,” Jerry nói trong khi tóm lấy bàn phím từ tay tôi. “Nhưng để tao chỉ cho mày, tao khoái làm cách đó như thế nào”.

public void testFileDoesNotExist() throws Exception {

c.setFilename(&quot;thisFileDoesNotExist&quot;);

boolean prepared = c.prepareFile();

assertEquals(false, prepared);

}

“Mày thấy không?” gã giảng giải. “Tao muốn đánh giá mỗi đối số của dòng lệnh trong hàm của chính nó thay vì trộn cả mớ mã phân tách và đánh giá 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 kiểm thử 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;

}

}

Toàn bộ các kiểm thử đều đạt. Jerry nhìn tôi rồi ngó sang bàn phím. Hiển nhiên gã muốn “lái” nó rồi. 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 người gã rõ ràng đang gầm rú.

public void testCountBytesInFile() throws Exception {

File f = new File(&quot;testFile&quot;);

FileOutputStream stream = new FileOutputStream(f);

stream.write(&quot;some text&quot;.getBytes());

stream.close();

c.setFilename(&quot;testFile&quot;);

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 prepareFile() để lấy độ dài của tệp tin? 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 tệp tin hiện có.”

“Mình cần nó để làm gì kia chớ?” tôi hỏi.

“Chúng ta sẽ phải gửi nội dung của tệp tin thông qua socket đến server, phải không?” Jerry hỏi.

“Vâng.”

“Và chúng ta cần biết sẽ gửi bao nhiêu chữ,” gã kiên nhẫn giải thích.

“Hừm… có lẽ,” tôi miễn cưỡng trả lời.

“Tin tao đi,” gã mỉm cười. “xét cho cùng thì tao là người hướng đạo cơ mà.”

“OK, khỏi nói đến chuyện ấy,” tôi trả lời một cách thiếu kiên nhẫn. “Tạo sao ông lại tạo tệp tin trong kiểm thử kia chớ? sao ông không giữ tệp tin 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 kiểm thử. Bất cứ khi nào có thể được, tao để cho mấy cái kiểm thử 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 thích thú gì với mấy thứ độ dài của tệp tin kia.”

“Nhớ đó. Mày sẽ thấy!”

Tôi lấy bàn phím và bắt đầu làm viết mã để vượt qua kiểm thử. Trong khi tôi gõ phím, tôi thấy hơi lạ vì tôi đang viết mã sản xuất trong khi thiết kế là của Jerry – nhưng những gì Jerry làm chỉ là viết những kiểm thử nhỏ. Bạn có thể thực sự xác định một thiết kế bằng cách viết những kiểm thử 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;

}

}

Kiểm thử 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);

}

Tôi có thể vượt qua nó mà chả gặp mấy khó khăn:

public boolean connect() {

try {

Socket s = new Socket(&quot;localhost&quot;, 9000);

return true;

} catch (IOException e) {

}

return false;

}

“Tuyệt!” Jerry nói. “Mình nghỉ 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 kiểm tra mấy thứ đó!”

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

**Bài tiếp:** [Thợ lành nghề #12: Ba dòng mã xấu xí (SMCRemote – phần 2)](https://codegym.vn/blog/2018/08/10/tho-lanh-nghe-12-ba-dong-ma-xau-xi-smcremote-phan-2/)

**Bài trước:** [Thợ lành nghề #10: Những thread lửng lơ (Vòng lặp không hạn chế)](https://codegym.vn/blog/2018/08/08/tho-lanh-nghe-10-nhung-thread-lung-lo-vong-lap-khong-han-che/)

**Tác giả:** Robert C. Martin

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