**Thợ lành nghề #7: Một lần không đủ (Dịch vụ Socket 2)**

Ngày 15 tháng 10 năm 2002.

*Lần trước Alphonse và Jerry khởi đầu trên một khung làm việc Java đơn giản hỗ trợ dịch vụ socket. Kiểm thử thứ nhất của họ vạch ra trường hợp dồn đuổi (race condition) mà họ đã giải quyết xong. Chuỗi kiểm thử đơn vị hiện tại được trình bày 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[]{&quot;TestSocketServer&quot;});

}

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(&quot;localhost&quot;, port);

try {

Thread.sleep(100);

} catch (InterruptedException e) {

}

s.close();

} catch (IOException e) {

fail(&quot;could not connect&quot;);

}

}

}

**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 tục với **SocketService**.

“Chúng ta đã chứng minh được là 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 một kiểm thử như sau:

public void testManyConnections() throws Exception {

SocketService ss = new SocketService();

ss.serve(999);

for (int i = 0; i &lt; 10; i++) {

connect(999);

}

ss.close();

assertEquals(10, ss.connections());

}

“OK, kiểm thử này hỏng.” Tôi nói.

“Nó nên như thế”. Jerry đáp. “Cái SocketService chỉ gọi phương thức accept một lần. Chúng ta cần đặt lời gọi đó vào một vòng lặp.”

“Khi nào vòng lặp đó chấm dứt?” Tôi hỏi.

Jerry nghĩ ngợi một lát và nói: “Khi chúng ta gọi phương thức close của SocketService.”

“Như thế này chăng?” Và tôi điều chỉnh 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 kiểm thử 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.”

“Ừ, đổi nó đi.” Jerry nói. “Mình hãy buộc **SocketService** gửi thông điệp “Hello” mỗi khi chúng ta truy cập đến nó.”

Tôi không quan tâm tới điều đó. Tôi nói: “Tại sao mình làm bẩn **SocketService** bằng thông điệp “Hello” chỉ để thoả mãn cái kiểm thử của mình? **SocketService** có thể gửi thông điệp thì tốt nhưng mình không muốn thông điệp này là một phần mã nguồn của **SocketService**!”

“Ðúng thế!” Jerry đồng ý. “Mình muốn thông điệp được chỉ định và xác thực do cái kiểm thử.”

“Mình làm sao đây?” Tôi hỏi.

Jerry mỉm cười đáp: “Chúng ta dùng mẫu thiết kế **Mock Object**. 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 kiểm thử ứng dụng cái interface đó dùng để gửi thông điệp “Hello”. Sau đó, mình sẽ có cái kiểm thử dùng để đọc thông điệp từ socket của máy khách (client) và xác thực thông tin được gửi đi một cách đúng đắn.”

Tôi chẳng biết mẫu thiết **Mock Object** 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 phím và bắt đầu gõ.

“Ðầu tiên chúng ta viết cái kiểm thử.”

public void testSendMessage() throws Exception {

SocketService ss = new SocketService();

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

Socket s = new Socket(&quot;localhost&quot;, 999);

InputStream is = s.getInputStream();

InputStreamReader isr = new InputStreamReader(is);

BufferedReader br = new BufferedReader(isr);

String answer = br.readLine();

s.close();

assertEquals(&quot;Hello&quot;, answer);

}

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

“Hay lắm!” Jerry thốt lên. “Ðiều đó có nghĩa là chúng ta cần tái cấu trúc những kiểm thử khác trước khi tiếp tục.”

“Nhưng các dịch vụ trong hai kiểm thử kia chẳng làm gì hết.” Tôi ý kiến.

“Tất nhiên là chúng có làm gì đó – 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. “Ðầu tiên chúng ta đổi hai cái kiểm thử và thêm biến số **connections** vào kiểm thử.”

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 &lt; 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 tạo nó trong hàm tạo của **TestSocketServer** bằng một lớp lồng nặc danh 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[]{&quot;TestSocketServer&quot;});

}

public TestSocketServer(String name) {

super(name);

connectionCounter = new SocketServer() {

public void serve(Socket s) {

connections++;

}

};

}

...

“Cuối cùng, chúng ta làm chúng có thể biên dịch trọn bộ bằng cách thêm đối số phụ vào phương thức **serve**của **SocketService** và chuyển cái kiểm thử mới thành chú thích (để nó khỏi chạy).”

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

...

}

“OK, tôi biết ý ông rồi.” Tôi nói. “Hai cái kiểm thử cũ lúc này hẳn phải hỏng bởi lẽ **SocketService** không bao giờ gọi phương thức serve của đối số **SocketServer** của nó.”

Tất nhiên các kiểm thử đã hỏng vì chính lý do ấy.

Tôi biết phải làm gì tiếp. Tôi vớ lấy bàn phím 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();

}

...

Ðoạn mã này làm các kiểm thử chạy được.

“Hay lắm!” Jerry nói. “Bây giờ chúng ta phải làm cho cái kiểm thử mới chạy.”

Thế nên tôi bỏ chú thích cho đoạn kiểm thử và biên dịch nó. Nó “la làng” trong phần **HelloServer**.

“Ô, đúng rồi. Mình phải cài đặt **HelloServer**. Nó sẽ phun ra chữ “hello” từ socket, phải không?”

“Ðúng thế.” Jerry xác nhận.

Thế rồi tôi viết lớp mới trong tệp **TestSocketServer.java** như sau

class HelloServer implements SocketServer {

public void serve(Socket s) {

try {

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

ps.println(&quot;Hello&quot;);

} catch (IOException e) {

}

}

}

Mọi kiểm thử đều ổn.

“Cũng dễ thôi.” Jerry nói.

“Ừ. Phần mẫu thiết kế Mock Object khá hữu dụng. Nó cho phép ta duy trì các mã dùng để kiểm thử trong kế hoạch kiểm thử. **SocketService** không biết gì cả.”

“Còn hữu dụng hơn thế.” Jerry trả lời. “Các server thật cũng sẽ ứng dụng interface **SocketServer**.”

“Tôi biết.” Tôi trả lời. “Thật lý thú khi thấy từ nhu cầu tạo ra một kiểm thử đơn vị đưa mình đến chỗ tạo ra một thiết kế hữu dụng một cách tổng quát.”

“Ðiều này thường xảy ra mà.” Jerry nói. “Kiểm thử là người dùng đó. Nhu cầu dùng kiểm thử 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 mẫu thiết kế này cho phép chúng ta thay thế bản nháp của chuyện kiểm thử vào mã nguồn ứng dụng thật.”

“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.”

“Ðồng 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();

}

}

**Bài tiếp:** [Thợ lành nghề #8: Kiểm thử là một dạng tài liệu (Dịch vụ Socket 3)](https://codegym.vn/blog/2018/08/06/tho-lanh-nghe-8-kiem-thu-la-mot-dang-tai-lieu-dich-vu-socket-3/)

**Bài trước:** [Thợ lành nghề #6: Một lần không đủ (Dịch vụ Socket 1)](https://codegym.vn/blog/tho-lanh-nghe-6-mot-lan-khong-du-dich-vu-socket-1/)

**Tác giả:** Robert C. Martin

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