**Thợ lành nghề #16: Lịch thiệp quá mức (SMCRemote – phần 6)**

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 nay 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 kiểm thử. Lúc đầu nàng có vẻ thỏa mãn nhưng rồi nàng nói, “FileCarrier đọc tệp tin 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 làm một phần của chuỗi. Hơn nữa, tệp tin 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 tệp tin mỗi lần một dòng và chuyên chở những tệp tin 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 kiểm thử có thể chạy. Khi mọi thứ đâu vào đó, tôi tái cấu trúc lớp 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 tệp tin một cách cần mẫn. Tôi muốn xem thêm vài cái kiểm thử.”

Lúc này nàng hết sức lịch thiệp. Một lần nữa, tôi dựng đoạn mã từng phần một, giữ cho các kiểm thử vẫn chạy được trong khi thay đổi mã nguồn. Tôi tái cấu trúc 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 tệp tin nguyên thuỷ được đặt tên lại và tệp tin mới được tạo ra. Không có điều gì có thể nghi ngờ rằng FileCarrier tạo tệp tin ở đây. Cũng không có cách nào tệp tin cũ bị bỏ rơi. Nhưng tôi chưa thấy phương thức 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 kiểm thử cho phương thức fileAreTheSame. Đầu tiên, tôi viết một cái kiểm thử chứng minh phương thức này làm việc ngon lành cho hai tệp tin như nhau. Rồi tôi viết một kiểm thử khác chứng minh hai tệp tin khác nhau không mang lại kết quả so sánh bằng nhau. Tôi viết tiếp thêm một cái kiểm thử nữa để chứng minh rằng nếu tệp tin này là tiền tố (prefix) của tệp tin kia thì sẽ không thể so sánh.

Kết cục tôi viết tổng cộng năm trường hợp kiểm thử khác nhau và chúng có cả lô mã trùng lặp: Mỗi kiểm thử viết hai tệp tin. Mỗi kiểm thử so sánh hai tệp tin. Mỗi kiểm thử xoá hai tệp tin. Để loại trừ phần trùng hợp này, tôi dùng mẫu thiết kề Template Method. Tôi dời trọn bộ các phần mã chung vào trong một lớp trừu tượng 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 tệp tin 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, filesAreTheSame(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 sử 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.

**Bài tiếp:** [Thợ lành nghề #17: Gọi bảo kê (SMCRemote – phần 7)](https://codegym.vn/blog/2018/08/14/tho-lanh-nghe-17-goi-bao-ke-smcremote-phan-7/)

**Bài trước:** [Thợ lành nghề #15: Ếch là Bê (SMCRemote – phần 5)](https://codegym.vn/blog/2018/08/13/tho-lanh-nghe-15-ech-la-be-smcremote-phan-5/)

**Tác giả:** Robert C. Martin

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