The Craftsman: 21 SMCRemote Part XI Patchwork.

Robert C. Martin
14 December 2003

...Continued from last month. You can download last month's code from:

www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_20_Backslide.zip

---

"Well now, Alphonse Dear, where do you think we should run the compile?"

I was still reeling from having just deleted all that code I wrote, so I missed the question.

"I'm sorry Jean, what did you say?"

"Well, we can't just run the SMC compiler any old place, can we dear? No, we've got to run it in a new directory. We have to copy all the incoming files into that directory, run the compile, and then send all the resulting files back to the caller."

"How many resulting files are there?" I asked. "I thought that SMC just produced a `.java` file."

"The Java code generator does dear. But the C++ code generator creates a `.h` and a `.cc` file. Other generators might produce other kinds of files. I'm afraid that it's a bit unpredictable. Anyway, dear, we want the compiler to run in a clean environment, and that means an empty directory with just the input `.sm` file present."

I thought about this for a second, and realized that she was right. I was starting to be very glad that I had deleted that old code. "So we should have a special directory to run the compiles in?"

"Oh, more than that dear. I think we should create a working directory as soon as we receive a `CompileFileTransaction`. Then we should write the input file into that directory. It'll be the only file there, and so things will be nice and clean, don't you agree? And then we can run the compiler, delete the input file, and gather up any remaining files and put them into the `CompilerResultsTransaction` for shipment back to the client. Oh my, this sounds like such *fun*. Why don't you write a little unit test that makes sure we can create a new directory, dear?"

I took the keyboard and started to type. While I was typing, Jean pulled a stack of brightly colored 6" cloth squares from her bag and started to sew them together.

```
  public void testMakeWorkingDirectory() throws Exception {
    File workingDirectory = SMCRemoteServer.makeWorkingDirectory();
    assertTrue(workingDirectory.exists());
    assertTrue(workingDirectory.isDirectory());
  }
```

"Oh, that's fine dear. First you create the directory, and then you make sure that it exists, and that it is, in fact, a directory. Now make that fine test pass."

I know Jean doesn't mean to be condescending, but every word grates on my nerves. I squared my shoulders and wrote the code to make the test pass. First I clicked on the missing function and had my IDE add it for me. Then I filled it in with the following code.

```
public static File makeWorkingDirectory() {
  File workingDirectory = new File("workingDirectory");
  workingDirectory.mkdir();
  return workingDirectory;
}
```

This code passed the test on the first try, of course, so I sat back and waited for Jean's next instruction.

"What are you waiting for dear?"

"I'm waiting for you to tell me what to do next."

"Are you? My goodness, I'm just a foolish, talkative old lady. Don't you have any ideas of your own? A bright young man such as yourself, you must be full of ideas. What do *you* think we should do next?

This was even more condescending than before. Jasmine and Jerry seemed to think that it was some kind of honor working with Jean; but I was finding it pretty painful.

"We should probably delete the directory when we are done with it." I said grumpily.

"I think that's a splendid idea. We'll have to make sure that the entire directory is deleted along with any files inside it."

So I wrote the following unit test while Jean continued sewing squares together.

```
public void testDeleteWorkingDirectory() throws Exception {
  File workingDirectory = SMCRemoteServer.makeWorkingDirectory();
  File someFile = new File(workingDirectory, "someFile");
  someFile.createNewFile();
  assertTrue(someFile.exists());
  SMCRemoteServer.deleteWorkingDirectory(workingDirectory);
  assertFalse(someFile.exists());
  assertFalse(workingDirectory.exists());
}
```

Jean looked up from her sewing and said: "Very nice, Alphonse. That's just what we want."

So then I wrote the following code:

```
public static void deleteWorkingDirectory(File workingDirectory) {
  workingDirectory.delete();
}
```

But this didn't pass the test. The test complained that `someFile` still existed.

"I think you have to delete all the files in the directory before you can delete the directory itself, Alphonse. I've never understood why they made that rule. I think it would be better if they believed you when you said you wanted the directory deleted. Oh well, I guess you'll have to loop through all the files and delete them one by one, dear."

So I wrote the following:

```
public static void deleteWorkingDirectory(File workingDirectory) {
  File[] files = workingDirectory.listFiles();
  for (int i = 0; i < files.length; i++) {
    files[i].delete();
  }
  workingDirectory.delete();
}
```

This made the tests pass. However I didn't like it. "Jean, this works, but what if there are subdirectories?"

"Yes, dear, we can't be sure that one of the SMC code generators won't create a subdirectory or two. I think you'd better augment the test to handle that case."

So I changed the test as follows:

```
public void testDeleteWorkingDirectory() throws Exception {
  File workingDirectory = SMCRemoteServer.makeWorkingDirectory();
  File someFile = new File(workingDirectory, "someFile");
  someFile.createNewFile();
  assertTrue(someFile.exists());

  File subdirectory = new File(workingDirectory, "subdirectory");
  subdirectory.mkdir();
  File subFile = new File(subdirectory, "subfile");
  subFile.createNewFile();
  assertTrue(subFile.exists());

  SMCRemoteServer.deleteWorkingDirectory(workingDirectory);
  assertFalse(someFile.exists());
  assertFalse(subFile.exists());
  assertFalse(subdirectory.exists());
  assertFalse(workingDirectory.exists());
}
```

Now the test failed as expected. The `subFile` was not getting deleted. So next I changed the code to make the test pass.

```
public static void deleteWorkingDirectory(File workingDirectory) {
  File[] files = workingDirectory.listFiles();
  for (int i = 0; i < files.length; i++) {
    if (files[i].isDirectory())
      deleteWorkingDirectory(files[i]);
    files[i].delete();
  }
  workingDirectory.delete();
}
```

"Oh that's just fine, dear, just fine. Very nicely done. Now, what next Alphonse?"

Something had been nagging at me, so I said: "Jean, this is a server isn't it?"

"Yes, dear. It waits on a socket for our client code to connect to it."

"How many clients might there be?"

"Oh, there could be dozens, perhaps more."

"Then if we get two clients connecting at the same time, we'll try to create the subdirectory twice, won't we?"

"Why yes, dear, I suppose that's true. That could cause quite a bit of trouble for us couldn't it. We'd have more than one session copying files into the directory, running compiles in the directory, copying files out of the directory, and deleting the directory too. That would be terrible wouldn't it? How do you think we should solve this Alphonse?"

"What if each working directory had a unique name? That way each of the incoming connections would have it's own directory to work in."

"That's just a superb idea, Alphonse. Why don't you write the test cases for that?"

I thought it was a superb idea too, so I wrote the following test.

```
public void testWorkingDirectoriesAreUnique() throws Exception {
  File wd1 = SMCRemoteServer.makeWorkingDirectory();
```

```
    File wd2 = SMCRemoteServer.makeWorkingDirectory();
    assertFalse(wd1.getName().equals(wd2.getName()));
  }
```

This test failed as expected. To make it pass I needed some way to create unique names. "I suppose I could use the time of day to generate a unique name." I said more to myself than to Jean.

"Yes, why don't you try that dear?"

So I wrote the following code.

```
  public static File makeWorkingDirectory() {
    File workingDirectory = new File(makeUniqueWorkingDirectoryName());
    workingDirectory.mkdir();
    return workingDirectory;
  }

  private static String makeUniqueWorkingDirectoryName() {
    return "workingDirectory" + System.currentTimeMillis();
  }
```

The test passed, but I saw a funny look in Jean's face. So I hit the test button again. She nodded while the test passed again. I hit the test button several more times, and sure enough there was an occasional failure.

"Yeah," I said, "if the two calls to `makeUniqueWorkingDirectoryName` are too close together, then `currentTimeMillis` returns the same time and the names aren't unique."

"Oh dear." said Jean.

"I suppose I could just use a static integer and increment for each name, but then the names would start over each time we restarted the server. That might cause collisions too."

"Good thinking dear." said Jean.

"Why don't we use both techniques." and I stopped talking and started typing.

```
public class SMCRemoteServer {
...
  private static int workingDirectoryIndex = 0;
...
  private static String makeUniqueWorkingDirectoryName() {
    return "workingDirectory" +
           System.currentTimeMillis() + "_" +
           workingDirectoryIndex++;
  }
...

}
```

Now the tests worked repeatedly and I was quite satisfied with myself. I was certain Jean would want a break soon, and we had gotten something done. On the other hand, what we got done was something very different from the code that I deleted before Jean came in.

"Alphonse."

"Yes, Jean."

"Would you mind deleting all those left over working directories dear? They're causing quite a big of clutter in our development directory. I just hate clutter, don't you? I hate it in my code, I hate it in my room, and I hate it in my directories."

I looked at the directory and noticed that there were dozens and dozens of left over working directories. I hung my head as I realized that my tests had not been cleaning up after themselves. I sheepishly made the required changes.

```
public void testMakeWorkingDirectory() throws Exception {
  File workingDirectory = SMCRemoteServer.makeWorkingDirectory();
  assertTrue(workingDirectory.exists());
  assertTrue(workingDirectory.isDirectory());
  SMCRemoteServer.deleteWorkingDirectory(workingDirectory);
}

public void testWorkingDirectoriesAreUnique() throws Exception {
  File wd1 = SMCRemoteServer.makeWorkingDirectory();
  File wd2 = SMCRemoteServer.makeWorkingDirectory();
  assertFalse(wd1.getName().equals(wd2.getName()));
  SMCRemoteServer.deleteWorkingDirectory(wd1);
  SMCRemoteServer.deleteWorkingDirectory(wd2);
}
```

"That's much better dear. Thank you. Now, I think it's time for a break, don't you?"

"Sounds good to me."

Jean had sewn 16 squares together into a 4X4 grid.

To be continued...

The code that Alphonse and Jean finished can be retrieved from:
www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_21_Patchwork.zip