The Craftsman: 15
SMCRemote Part V
Ess Are Pee

Robert C. Martin
18 June, 2003

...Continued from last month. See <<link>> for last month's article, and the code we were working on. You can download that code from:

www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_14_SMCRemote_IV_Transactions.zip

---

The last thing I saw, before the door slid shut, was Jasmine's long black hair swaying and bouncing to the rhythm of her purposeful stride. As her saturating presence drained from the room, I felt my lungs release the breath I hadn't known they were holding. My eyes lost their focus, and for several quasi-conscious minutes I just sat and blindly gazed at the blur that was the door.

"Whew!" I said to myself. "Working with Jasmine is *not* going to be easy."

"Easy or not, " I replied to myself, *"she asked you to get that file written and to clean up the code. So you'd better get cracking."*

I sighed. "Agreed, agreed." I agreed with myself. "I can just imagine what Jasmine would say if I didn't have something to show her. She'd say: *'Some Hotshot you turned out to be! What did you do, just sit here the whole time doing nothing?'*"

"Okay, Jasmine, okay. So, what should we do first?"

"Well, Hotshot, we have to get the `CompilerResultsTransaction` *to carry the contents of a file from the server to the client; and then write that file on the client."*

"Oh right." I said. "The compiler will create an output file on the server, and we have to move it to the client. This is just what we did in the `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;
  }
}
```

"We opened and read the file in the `compileFile` method, and then constructed and passed the file name and the array of chars into the constructor to the `CompileFileTransaction`."

```
public boolean compileFile() {
    char buffer[] = new char[(int) itsFileLength];
    try {
      fileReader.read(buffer);
      CompileFileTransaction cft =
        new CompileFileTransaction(itsFilename, buffer);
```

"Right, Hotshot, that's just what we did. Is there something you don't like about that?"

"I don't like having to write the same code twice. Jerry made a big point out of not duplicating code."

"Jerry's not exactly the sharpest knife in the drawer, Hotshot".

"Maybe not, but I think he was right about that. So I think I want to write a class that carries a file across the socket."

"Something like a `FileCarrier`?"

"Yeah, that's a good name for it!"

"OK, Hottie, write a test for it."

"Hottie? -- Er, OK. How about this?"

```
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);
  }
}
```

"Yes, very good, Hot Stuff, keep going."

"OK, beautiful. Here are the two utility functions..."

```
  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 TESTFILE,
                          final String TESTSTRING) throws IOException {
    PrintWriter writer = new PrintWriter(new FileWriter(TESTFILE));
    writer.println(TESTSTRING);
    writer.close();
  }
```

"...and here is the stubbed implementation of `FileCarrier` that will make this test compile and fail.

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

  public void write() {
  }
}
```

"So now, Jazzy-wazzy, all we have to do to make the test pass is read the file in the constructor and write it in the `write()` function."

"Not yet, over-temp, first run the test to make sure it will fail."

"Jay-girl, there's no implementation in `FileCarrier`. Of course it's going to fail.

"Well, exotherm, you and I both know that. But does the program?"

"I love it when you make me run a test! OK, here goes."

*** the test passes***

"Huh? What? How can that be? Do you understand that Jazz?"

"Gosh, boiling point, I sure don't. How can the test pass when there's nothing in `FileCarrier` *to..."*

"Oh. Egad. Are we dumb or what. We never deleted the test file."

"Ah, yes, I see that now, Mr. tepid breath. So why don't you fix that?"

"OK, here."

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

"OK, great, now it fails. But -- latent-heat -- can you make it pass?"

"Sure I can, JJ, just watch me!"

```
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 Ha! electron-volt, you are cooking now!"

"Why, thanks! Gorgeous! But you haven't seen anything yet. Watch how I integrate the `FileCarrier` into the `CompileFileTransaction`! That should turn your head."

```
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();
  }
}
```

"And now I'll change the `compileFile` function to use the new `CompileFileTransaction`!"

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

"And now I'll run all the tests and ... see? They all pass!"

"Oh, fever man, stop it, stop it, you're making me crazy!"

"It's all part of my plan, sweet-eyes, all part of my plan. Now, watch with bated breath as I put the `FileCarrier` into the `CompilerResultsTransaction`!"

"Ooooh!"

```
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();
  }
}
```

"Gasp!"

"That's right! And look how masterfully I change the test to use the new transaction!"

```
  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();
      }
    }
  }
```

"More, Alphonse, Please, More!"

You want more Jasmine? I can give you more. Lots more. You see Jasmine, I *know* about objects. Oh yes, I know about them. `FileCarrier` is an *Object* Jasmine. Do you see how it can be used in more than one place, Jasmine? Do you see how it encapsulates a single responsibility, Jasmine? Do you? Do you know the **Single Responsibility Principle** Jasmine? Have you heard of it? Have you studied it? Well I have. Do you know what it says, Jasmine? It says that a class should have one and only one reason to change, Jasmine. It says that all the functions and variables of a class should work together towards a single goal, Jasmine. It says that a class should not try to accomplish more than one goal...Are you *listening* Jasmine?

"Yes, Alphonse. I'm listening. Please, don't stop!"

"You'd better not ask me to stop! You see, Jasmine, those of us in the know call this principle the SRP. That's ESS ARE PEE Jasmine.

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

"Remember the `compileFile` function Jasmine, remember how it used to read the file and pass a `char` array into the `CompileFileTransaction`? You asked me what I didn't LIKE about that function. Well I'll TELL you what I didn't LIKE about it, Jasmine; it was VIOLATING the SRP! It had TWO reasons to change, instead of ONE. It depended BOTH on the details of reading a file, AND on the policy of building and sending transactions. That's TOO MUCH RESPONSIBILITY Jasmine."

"You're scaring me, Alphonse."

"You SHOULD be scared, Jasmine. I'm..."

"Oh! Alphonse!"

~ ~ ~

Then several things happened at once:

1. I noticed that the door was open.
2. I noticed that the chair next to me was empty.
3. I heard the echoes of my falsetto imitation of Jasmine still reverberating off the walls.
4. I saw Jasmine standing in the doorway. Her eyes were cold steel.

To be continued...maybe.

The code that Alphonse and Alphonse finished can be retrieved from:

www.objectmentor.com/resources/articles/CraftsmanCode/Craftsman_15_SMCRemote_V_EssArePee.zip