The Craftsman: Three.

Robert C. Martin
13 February 2002
This chapter is derived from a chapter of Principles, Patterns, and Practices of Agile Software Development, *Robert C. Martin, Prentice Hall, 2002.*

Previously Alphonse, the apprentice, was asked by Jerry, the Jouneyman, to write a program to generate prime numbers using the sieve of Eratosthenes. Jerry has been reviewing the code and helping Alphonse refactor it. He is not happy with Alphonse's work. In the last column Alphonse had just grabbed the keyboard and done a refactoring that he thought Jerry might approve of...

...Jerry was just barely nodding his head. Did he actually like what I did?

Next Jerry made pass over the whole program, reading it from beginning to end, rather like he was reading a geometric proof. He told me that this was a real important step. "So far", he said, "We've been refactoring fragments. Now we want to see if the whole program hangs together as a readable whole."

I asked: "Jerry, do you do this with your own code too?"

Jerry scowled: "We work as a team around here, so there is no code I call my own. Do you consider this code yours now?"

"Not anymore." I said, meekly. "You've had a big influence on it."

"We *both* have." he said, "And that's the way that Mr. C likes it. He doesn't want any single person owning code. But to answer your question: Yes. We all practice this kind of refactoring and code clean up around here. It's Mr. C's way."

During the read-through, Jerry realized that he didn't like the name `initialize-ArrayOfIntegers`.

"What's being initialized is not, in fact, an array of integers;", he said, "it's an array of booleans. But `initializeArrayOfBooleans` is not an improvement. What we are really doing in this method is uncrossing all the relevant integers so that we can then cross out the multiples."

"Of course!" I said. So I grabbed the keyboard and changed the name to `uncross-IntegersUpTo`. I also realized that I didn't like the name `isCrossed` for the array of

booleans. So I changed it to `crossedOut`. The tests all still run. I was starting to enjoy this; but Jerry showed no sign of approval.

Then Jerry turned to me and asked me what I was smoking when I wrote all that `maxPrimeFactor` stuff. (See Listing 6.) At first I was taken aback. But as I looked the code and comments over I realized he had a point. Yikes, I felt stupid! The square root of the size of the array is not necessarily prime. That method did not calculate the maximum prime factor. The explanatory comment was just wrong. So I sheepishly rewrote the comment to better explain the rationale behind the square root, and renamed all the variables appropriately. The tests all still ran.

Listing 6

```
TestGeneratePrimes.java (Partial)
  private static int calcMaxPrimeFactor()
  {
    // We cross out all multiples of p, where p is prime.
    // Thus, all crossed out multiples have p and q for
    // factors.  If p > sqrt of the size of the array, then
    // q will never be greater than 1.  Thus p is the
    // largest prime factor in the array, and is also
    // the iteration limit.
    double maxPrimeFactor = Math.sqrt(isCrossed.length) + 1;
    return (int) maxPrimeFactor;
  }
```

"What the devil is that `+1` doing in there?" Jerry barked at me.

I gulped, looked at the code, and finally said: "I was afraid that a fractional square root would convert to an integer that was one too small to serve as the iteration limit."

"So you're going to litter the code with extra increments just because you are paranoid?" he asked. "That's silly, get rid of the increment and run the tests."

I did, and the tests all ran. I thought about this for a minute, because it made me nervous. But I decided that maybe the true iteration limit was the largest prime less than or equal to the square root of the size of the array.

"That last change makes me pretty nervous." I said to Jerry. "I understand the rationale behind the square root, but I've got a nagging feeling that there may be some corner cases that aren't being covered."

"OK," he grumbled. "So write another test that checks that."

"I suppose I could check that there are no multiples in any of the prime lists between 2 and 500."

"OK, if it'll make you feel better, try that." he said. He was clearly becomming impatient.

So I wrote the `testExhaustive` function shown in Listing 8. The new test passed, and my fears were allayed.

Then Jerry relented a bit. “It’s always good do know why something works;” said Jerry. “and it’s even better when you show you are right with a test.”

Then Jerry scrolled one more time through all the code and tests (shown in listings 7 and 8). He sat back, thinking for a minute, and said: “OK, I think we’re done. The code looks reasonably clean. I’ll show it to Mr. C.”

Then he looked me dead in the eye and said: “Remember this. From now on when you write a module, get help with it and keep it clean. If you hand in anything below those standards you won’t last long here.”

And with that, he strode off.

Listing 7

`PrimeGenerator.java (final)`

```
/**
 * This class Generates prime numbers up to a user specified
 * maximum.  The algorithm used is the Sieve of Eratosthenes.
 * Given an array of integers starting at 2:
 * Find the first uncrossed integer, and cross out all its
 * multiples.  Repeat until there are no more multiples
 * in the array.
 */

public class PrimeGenerator
{
  private static boolean[] crossedOut;
  private static int[] result;

  public static int[] generatePrimes(int maxValue)
  {
    if (maxValue < 2)
      return new int[0];
    else
    {
      uncrossIntegersUpTo(maxValue);
      crossOutMultiples();
      putUncrossedIntegersIntoResult();
      return result;
    }
  }

  private static void uncrossIntegersUpTo(int maxValue)
  {
    crossedOut = new boolean[maxValue + 1];
    for (int i = 2; i < crossedOut.length; i++)
      crossedOut[i] = false;
  }

  private static void crossOutMultiples()
  {
```

```
    int limit = determineIterationLimit();
    for (int i = 2; i <= limit; i++)
      if (notCrossed(i))
        crossOutMultiplesOf(i);
  }

  private static int determineIterationLimit()
  {
    // Every multiple in the array has a prime factor that
    // is less than or equal to the sqrt of the array size,
    // so we don't have to cross out multiples of numbers
    // larger than that root.
    double iterationLimit = Math.sqrt(crossedOut.length);
    return (int) iterationLimit;
  }

  private static void crossOutMultiplesOf(int i)
  {
    for (int multiple = 2*i;
         multiple < crossedOut.length;
         multiple += i)
      crossedOut[multiple] = true;
  }

  private static boolean notCrossed(int i)
  {
    return crossedOut[i] == false;
  }

  private static void putUncrossedIntegersIntoResult()
  {
    result = new int[numberOfUncrossedIntegers()];
    for (int j = 0, i = 2; i < crossedOut.length; i++)
      if (notCrossed(i))
        result[j++] = i;
  }

  private static int numberOfUncrossedIntegers()
  {
    int count = 0;
    for (int i = 2; i < crossedOut.length; i++)
      if (notCrossed(i))
        count++;

    return count;
  }
}
```

Listing 8
TestGeneratePrimes.java (final)

```
import junit.framework.*;

public class TestGeneratePrimes extends TestCase
{
  public static void main(String args[])
  {
      junit.swingui.TestRunner.main(
        new String[] {"TestGeneratePrimes"});
  }
  public TestGeneratePrimes(String name)
  {
    super(name);
  }

  public void testPrimes()
  {
    int[] nullArray = PrimeGenerator.generatePrimes(0);
    assertEquals(nullArray.length, 0);

    int[] minArray = PrimeGenerator.generatePrimes(2);
    assertEquals(minArray.length, 1);
    assertEquals(minArray[0], 2);

    int[] threeArray = PrimeGenerator.generatePrimes(3);
    assertEquals(threeArray.length, 2);
    assertEquals(threeArray[0], 2);
    assertEquals(threeArray[1], 3);

    int[] centArray = PrimeGenerator.generatePrimes(100);
    assertEquals(centArray.length, 25);
    assertEquals(centArray[24], 97);
  }

  public void testExhaustive()
  {
    for (int i = 2; i<500; i++)
        verifyPrimeList(PrimeGenerator.generatePrimes(i));
  }

  private void verifyPrimeList(int[] list)
  {
    for (int i=0; i<list.length; i++)
        verifyPrime(list[i]);
  }

  private void verifyPrime(int n)
  {
    for (int factor=2; factor<n; factor++)
      assert(n%factor != 0);
  }
}
```

What a disasater! I thought sure that my original solution had been top-notch. In some ways I still feel that way. I had tried to show off my brilliance, but I guess Mr. C values collaboration and clarity more than individual brilliance.

I had to admit that the program reads much better than it did at the start. It also works a bit better. I was pretty pleased with the outcome. Also, in spite of Jerry's gruff attitude, it had been *fun* working with him. I had learned a lot.

Still, I was pretty discouraged with my performance. I don't think the folks here are going to like me very much. I'm not sure they'll ever think I'm good enough for them. This is going to be a lot harder than I thought.

To Be Continued Next Month