The Craftsman: Two.

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.*

Dear Diary,

13 February 2002,

Today was a disaster – I really messed it up. I wanted so much to impress the journeymen here, but all I did was make a mess.

It was my first day on the job as an apprentice to Mr. C. I was lucky to get this apprenticeship. Mr. C is a well recognized master of software development. The competition for this position was fierce. Mr. C's apprentices often become journeymen in high demand. It *means* something to have worked with Mr. C. ...

In last month's column Alphonse, the apprentice, was asked by Jerry, the Jouneyman, to write a program to generate prime numbers using the sieve of Eratosthenes. Jerry, noting that Alponse implemented the entire algorithm in a single gargantuan function has asked Alphonse to split it by separating the three concepts: inialization, generation, and output preparation;... but Alphonse doesn't know how to start...

He looked at me for awhile, clearly expecting me to do something. But finally he heaved a sigh, shook his head, and continued. "To expose these three concepts more clearly, I want you to extract them into three separate methods. Also get rid of all the unnecessary comments and pick a better name for the class. When you are done with that, make sure all the tests still run."

You can see what I did in Listing 3. I've marked my changes in bold, just like Martin Fowler does in his *Refactoring* book. I changed the name of the class to a noun, got rid of all the comments about Eratosthenes, and made three methods out of the three concepts in the `generatePrimes` function.

Extracting the three functions forced me to promote some of the variables of the function to static fields of the class. Jerry said that this made it much clearer which variables are local and which have wider influence.

Listing 3

PrimeGenerator.java, version 2

```
/**
 * 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 the first uncrossed integer exceeds
 * the square root of the maximum value.
 */
import java.util.*;

public class PrimeGenerator
{
  private static int s;
  private static boolean[] f;
  private static int[] primes;

  public static int[] generatePrimes(int maxValue)
  {
    if (maxValue < 2)
      return new int[0];
    else
    {
      initializeSieve(maxValue);
      sieve();
      loadPrimes();
      return primes;  // return the primes
    }
  }

  private static void loadPrimes()
  {
    int i;
    int j;

    // how many primes are there?
    int count = 0;
    for (i = 0; i < s; i++)
    {
      if (f[i])
        count++; // bump count.
    }

    primes = new int[count];

    // move the primes into the result
```

```
    for (i = 0, j = 0; i < s; i++)
    {
      if (f[i])             // if prime
        primes[j++] = i;
    }
  }

  private static void sieve()
  {
    int i;
    int j;
    for (i = 2; i < Math.sqrt(s) + 1; i++)
    {
      if (f[i])  // if i is uncrossed, cross out its multiples.
      {
        for (j = 2 * i; j < s; j += i)
          f[j] = false; // multiple is not prime
      }
    }
  }

  private static void initializeSieve(int maxValue)
  {
    // declarations
    s = maxValue + 1; // size of array
    f = new boolean[s];
    int i;

    // initialize array to true.
    for (i = 0; i < s; i++)
      f[i] = true;

    // get rid of known non-primes
    f[0] = f[1] = false;
  }
}
```

Jerry told me that this was a little messy, so he took the keyboard and showed me how to clean it up. Listing 4 shows what he did. First he got rid of the `s` variable in `initializeSieve` and replacing it with `f.length`. Then he changed the names of the three functions to something he said was a bit more expressive. Finally he rearranged the innards of `initializeArrayOfIntegers` (née `initializeSieve`) to be a little nicer to read. The tests all still ran.

Listing 4

`PrimeGenerator.java, version 3 (partial)`

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

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

private static void initializeArrayOfIntegers(int maxValue)
{
  f = new boolean[maxValue + 1];
  f[0] = f[1] = false;  //neither primes nor multiples.
  for (int i = 2; i < f.length; i++)
    f[i] = true;
}
```

I had to admit, this was a bit cleaner. I'd always thought it was a waste of time to give functions long descriptive names, but his changes really did make the code more readable.

Next Jerry pointed at `crossOutMultiples`. He said he thought the `if(f[i] == true)` statements could be made more readable. I thought about it for a minute. The intent of those statements was to check to see if `i` was uncrossed; so I changed the name of `f` to `unCrossed`.

Jerry said that this was better, but still wasn't pleased with it because it lead to double negatives like `unCrossed[i] = false`. So he changed the name of the array to `isCrossed` and changed the sense of all the booleans. Then he ran all the tests.

Jerry got rid of the initialization that set `isCrossed[0]` and `isCrossed[1]` to `true`. He said it was good enough to just made sure that no part of the function used the `isCrossed` array for indexes less than 2. The tests all still ran.

Jerry extracted the inner loop of the `crossOutMultiples` function and called it `crossOutMultiplesOf`. He said that statements like `if (isCrossed[i] == false)` were confusing so he created a function called `notCrossed` and changed the `if` statement to `if (notCrossed(i))`. Then he ran the tests.

Then Jerry asked me what that square root was all about. I spent a bit of time writing a comment that tried to explain why you only have to iterate up to the square root of the array size. I tried to emulate Jerry by extracting the calculation into a function where I could put the explanatory comment. In writing the comment I realized that the square root is the maximum prime factor of any of the integers in the array. So I chose that name

for the variables and functions that dealt with it. Finally, I made sure that the tests all still ran. The result of all these changes are shown in Listing 5.

Listing 5

PrimeGenerator.java version 4 (partial)

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

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

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

  private static void crossOutMultiples()
  {
    int maxPrimeFactor = calcMaxPrimeFactor();
    for (int i = 2; i <= maxPrimeFactor; i++)
      if (notCrossed(i))
        crossOutMultiplesOf(i);
  }

  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;
  }

  private static void crossOutMultiplesOf(int i)
```

```
    {
      for (int multiple = 2*i;
           multiple < isCrossed.length;
           multiple += i)
        isCrossed[multiple] = true;
    }

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

I was starting to get the hang of this so I took a look at the `putUncrossed-IntegersIntoResult` method. I saw that this method had two parts. The first counts the number of uncrossed integers in the array, and creates the result array of that size. The second moves the uncrossed integers into the result array. So, as you can see in Listing 6, I extracted the first part into its own function and did some miscellaneous cleanup. The tests all still ran. Jerry was just barely nodding his head. Did he actually like what I did?

Listing 6

`PrimeGenerator.java, version 5 (partial).`

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

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

      return count;
    }
```

To Be Continued Next Month