Activity 2-4 Refactoring: Renaming Variables and Extracting Classes

Some IntelliJ Keyboard Shortcuts

- *Refactor This* menu: <ctrl + alt + shift>−T
- *Show available intentions*: <alt>−ENTER

Getting Started

☐ Download primeGenerator.zip from the **Refactoring Activity 1-3** folder on the Moodle class page (right-click and save link to your directory for this class).

☐ In a terminal window, navigate to that directory and run: unzip primeGenerator.zip. This will create a directory called primeGenerator.

☐ Start IntelliJ and select: *File -> New -> Project from Existing Sources*.

☐ Navigate to and select the primeGenerator folder that you just unzipped. Follow the Next buttons to complete the process for creating the project.

☐ Verify that the project has the following three files:

- PrimePrinter.java
- PrintPrimesTest.java
- gold

☐ Open up the "PrintPrimesTest" file, and expand the "import" section by clicking the plus button next to the line.

☐ Place your cursor over the red "junit" text, then trigger the intention menu (alt-Enter) and select "add junit 4 to the classpath". All the red marks should now have disappeared.

☐ Create a git repository for the project from within IntelliJ: *VCS -> Enable Version Control Integration* then choose Git as the version control system.

☐ Choose *VCS -> Commit* to create the first commit. Select the topmost checkbox to include all the files in the commit, type a Commit Message of "Initial Commit" then hit the Commit button.

☐ Create a remote git repository for the project on GitHub from within IntelliJ (you will need to have an account): VCS −> Import into version control −> Share project on GitHub

☐ Right-click on "PrintPrimesTest" (either the filename or the tab) and select: Run 'PrintPrimesTest' from the given menu. This will verify that everything is setup properly. Ask for help if the test fails.

- ☐ Take a look at the PrimePrinter.java and its main method. This is the method we will be refactoring.
- ☐ Take a look at the "gold" file. This contains the output of our method; our method produces this printed result. The tests simply compare the function's output to this printout.

Step 1: Rename Variables

1. The first and easiest thing to fix are the variable names. The second and third columns of the table below give the variables used by the program and a description of what each variable represents. The first column gives a new-and-improved name for most of the variables.

Take a look through the list of variable names. Note that for some variables, it is how the name is formatted rather than the name itself that needed to be improved (e.g., camel-case instead of all-caps).

New Name	Current Name	Description
numPrimes	M	number of primes to generate and print
rowsPerPage	RR	number of rows to print per page
colsPerPage	CC	number of columns to print per page
	ORDMAX	Leave as is; we will come back to this one later
primes	P	list of primes
pageNumber	PAGENUMBER	current page number in the printout
pageOffset	PAGEOFFSET	offset into the primes array where the current page starts
rowOffset	ROWOFFSET	offset into the primes array where the elements in the current row start
col	C	current column in the printout
candidatePrime	J	candidate prime number
lastPrimeIndex	K	index into the prime array for the last computed prime
possiblyPrime	JPRIME	boolean that indicates whether the candidate number (J) is "possibly prime" (false means definitely not prime)
	ORD	Leave as is; we will come back to this one later
nextPrimeSquare	SQUARE	next possible prime square
	N	Leave as is; we will come back to this one later
multiples	MULT	an array of multiples

2. Rename each variable in PrimePrinter.java as follows:
    a. Click on the variable name.
    b. Use <ctrl+alt+shift>–T to open the **Refactor This** menu.
    c. Choose **Rename** (the variable will become outlined in red).

d. Type in the new name for the variable.
e. ENTER to make the change.

Notice that all occurrences of the original variable name have been renamed.

3. Use *Code -> Reformat Code* to automatically fix the indentation and spacing.
4. The last thing to do is to move the variable declarations closer to where the variables are first used. For each variable declaration, do the following:

   a. Click on a variable name
   b. Use <alt>–ENTER to show available intentions
   c. Select *Move declaration closer to usages* if that is one of the possible intentions in the list
      - Make sure to do numPrimes last, as the location of the other declarations limits our ability to move it earlier.
      - Rerun your tests to make sure they run

5. Commit changes to your local GitHub:

   a. *VCS -> Commit* to open the **Commit Changes** window
   b. Make sure all the files are checked, otherwise select them yourself.
   c. Add the commit message, "rename variables", and commit your changes.

Step 2: Partition Code into Basic Methods

Looking at the structure of the PrimePrinter code, we can see that it splits broadly into two parts:

- The first while loop, which seems to do be doing the work of computing prime numbers.
- The second while loop (after pageNumber and pageOffset are initialized), which seems to have to do with printing the numbers.

We want to isolate each loop by turning it into its own method. However, before doing that, we need to think about the variables being used. They fall broadly into three categories:

- Global constants that represent settings that the user might want to adjust: numPrimes, rowsPerPage, colsPerPage.
- variables used for the prime generation work: ORDMAX, primes, multiples, candidatePrime, lastPrimeIndex, ORD, nextPrimeSquare, and maybePrime.
- variables used for printing the primes: pageNumber, pageOffset, rowOffset, col.

We want the second set of variables to be local variables in a function called generatePrimes, which we will generate from the first while loop. We want the third set of variables to be local variables in a function called printNumbers, which we will generate from the second while loop.

Follow the steps below to refactor and extract the two main while loops into separate methods:

1. Extract the generatePrimes Method (Try 1)

   a. Select all the lines of code that make up the first main while loop.
   b. Use <ctrl+alt+shift>–T to open the **Refactor This** menu.
   c. Under **Extract** select **Method**.
   d. In the **Extract Method** window, give it the name "generatePrimes" and take a moment to look over the proposed parameters and return type.
   e. Click OK.
   f. *Run tests to verify that nothing has been broken.*

   After this refactor, the first while loop and all its code should now be down at the bottom of the file in a method called "generatePrimes". In its place should be the following function call (with possibly different parameter order):

   ```
   generatePrimes(numPrimes, primes, lastPrimeIndex, ORD,
           nextPrimeSquare, multiples, candidatePrime);
   ```

   Undo and redo the method extraction a few times until you can see this is what happens.

2. Extract the generatePrimes Method (Try 2)

   Although the above refactoring gave us the method we wanted, the variables that we want to be declared locally are instead being passed in as parameters. There are a couple ways to fix this, but the simplest is to redo the refactoring, this time making sure to include all the declarations for the variables we want to be local in the new method.

   a. Undo the extract method refactoring so that the while loop is back in main.
   b. Check the location of the variables that we want to be part of generatePrimes. They should be right above the first while loop, and the variables we want to be local to printNumbers should be directly above the second while loop. If not, move the variables to where they should be.
   c. Select the code of the first while loop again, but this time be sure to include all the variable declarations above it, **with the exception of numPrimes**.
   d. Repeat the extract-method refactoring.
   e. *Run tests to verify that nothing has been broken.*

   Your line calling generatePrimes should now look like the following:

   ```
   int[] primes = generatePrimes(numPrimes);
   ```

There are a couple things you should note about this second try at refactoring the first while loop:

- It resulted in a generatePrimes method with a much simpler signature that requires the passing of just a single parameter: numPrimes.
- The refactoring was smart enough when creating generatePrimes to have it return the the correct thing, the primes array.

3. Extract the printNumbers Method

   a. Select the code for the remaining while loop in main, including all the variable declarations after the call to generatePrimes.
   b. **Refactor This** (<ctrl+alt+shift>–T) and extract the method.
   c. *Run tests to verify that nothing has been broken.*

4. Commit your changes:

   a. *VCS -> Commit* to open the **Commit Changes** window
   b. Add the commit message, "extract generatePrimes and printNumbers methods", and commit your changes.

After the above refactoring, your main method should be short and clear as to what it aims to accomplish:

```
public static void main(String[] args) {
    final int numPrimes = 1000;
    int[] primes = generatePrimes(numPrimes);
    printNumbers(numPrimes, primes);
}
```

Step 3: Extracting Classes

The two methods created above are a good start: each handles a specific and disjoint set of variables. However, both methods are still quite large, and they deal with two mostly unrelated tasks. This suggests that the two methods really belong in two different classes. Let's use our two methods as the starting point for creating these classes.

1. Create the PrimeGenerator class:

   a. Select the generatePrimes call in main.
   b. **Refactor This** (<ctrl+alt+shift>–T) and under **Extract** select **Method Object**.
   c. Name the new class PrimeGenerator.
   d. Keep the visibility private and double check the signature before clicking OK.
   e. When asked, move the generatePrimes method to the extracted class.

   The new class was created as an inner class to the PrimePrinter class, but ideally we would like it to be its own class.

f. Select the name of the PrimeGenerator class from the first line of the class definition.

g. **Refactor This** (<ctrl+alt+shift>−T) and select **Move**.

h. Move the inner class PrimeGenerator to upper level.

i. When asked, add PrimeGenerator.java to the git repository.

Take a look at the newly created PrimeGenerator class. We do not need both the invoke and the generatePrimes methods. Currently invoke is just a wrapper around the call to generatePrimes. To clean this up, we will inline generatePrimes and then rename invoke.

j. Click on the first line (definition) of the generatePrimes method.

k. **Refactor This** (<ctrl+alt+shift>−T) and select **Inline**.

l. Accept the default, which is to "Inline all and remove the method".

All the code from generatePrimes should now be part of the invoke method, and the generatePrimes method should be gone.

n. *Refactor This* to rename invoke to generatePrimes.

o. Run the test to make sure everything is still working.

We are far from done with refactoring the PrimeGenerator class: It still consists of one large method, an we will need to do something about that. For now, though, let's move on to the printNumbers method.

2. Create the NumberPrinter class

   Back in PrimePrinter.java, take a moment to look at the parameters and variables at the top of printNumbers:

   - The colsPerPage and rowsPerPage variables feel like they should be part of the initialization of the new class: When you create a number-printer, you should be specifying how many rows and columns you want it to print.
   - The primes and numPrimes parameters feel like elements that should be provided to printNumbers when it is invoked.
   - The pageNumber and pageOffset will likely end up being internal variables to the new class.

   To get the variables and parameters where they need to be in the new class, we will first turn colsPerPage and rowsPerPage into parameters, and then extract a parameter object (instead of a method object).

   a. Select colsPerPage.

   b. **Refactor This** (<ctrl+alt+shift>−T) and extract it as a parameter.

   c. Hit Enter to keep the parameter name the same as the variable name.

   d. Repeat the two steps above to extract rowsPerPage as a parameter. At this point your printNumbers call should have four parameters in some order: numPrimes, primes, colsPerPage, rowsPerPage

e. Select the printNumbers method call or definition.
f. Open the **Refactor This** (<ctrl+alt+shift>−T) menu.
g. Under **Extract** select **Parameter Object**.
h. Name the new class NumberPrinter.
i. Under **Parameters to Extract unselect** numPrimes and primes.
j. Refactor.
k. Add the new file to the git repository when asked.

The printNumbers method should now have three parameters: numPrimes, primes, and numberPrinter, which is a NumberPrinter object. And the call to printNumbers contains in it the creation of this new class.

Now we'll change printNumbers to a method of the NumberPrinter class:

k. Select the printNumbers call.
l. **Refactor This** (<ctrl+alt+shift>−T) and select **Convert To Instance Method**.
m. Select the numberPrinter instance as the destination (and not the this/new NumberPrinter()..).

Take a look at the NumberPrinter class. IntelliJ has "encapsulated" the two fields behind getter methods, which we do not really need. Inline each of these getter methods one at a time to remove them:

m. Select the method name.
n. **Refactor This** (<ctrl+alt+shift>−T) and select **Inline**.
o. Inline all and remove the method.
p. Verify that all tests are still passing.

3. Go back to PrimePrinter.java. The program is now nice and simple – it initializes a few parameters, creates a prime number generator and calls it, and creates a number printer and calls it with the primes. However, there is a bit more refactoring we can do to clean it up even more.

   a. Select "new NumberPrinter(4, 50)" and **Refactor This** to extract a local variable called numberPrinter.
   b. Select "new PrimeGenerator(numPrimes)" and refactor to extract a local variable called primeGenerator.
   c. Select the primes array in its definition and refactor to inline the variable, moving the construction of the array so that it now occurs inline as part of the printNumbers method call.
   d. Use refactoring to rename the printNumbers method to just print, and the generatePrimes method to just generate.

4. One last time, run the tests to make sure that everything is still working.

5. Commit and push changes.

a. *VCS -> Commit* to open the **Commit Changes** window

b. Add the commit message, “refactor to create PrimeGenerator an NumberPrinter classes”, and commit your changes.

c. *VCS -> Git -> Push*

This activity continues in refactoring activity 2[1].

[1] activity2-5aRefactoringPrimesGeneratorPart2.html