

# Refactoring Activity 1

- Access the “Prime Generator” code from Moodle, and unzip it
- Start an IntelliJ project from the provided sources
- Run the PrintPrimesTest file, and make sure it passes.
- Take a look at the PrimePrinter file and its main method. This is the method we will refactor and clean up.

## Step 1: Variable renaming

We will start by giving better name for each variable. After reading the description of the variable here, choose a better name for it and use the “Rename” refactoring on the variable. Some variables may simply need to be written in camel-case instead of all-caps.

**M** This is the number of primes that we will generate and print

**RR** This is the number of rows to print per page

**CC** This is the number of columns to print per page

**ORDMAX** Leave as is, but make lower case (TODO: come back to it?)

**P** The list of primes

**PAGENUMBER** The current page number in the printout

**PAGEOFFSET** This is the offset into the primes array that the current page would be starting at

**ROWOFFSET** This is the offset into the primes array for the elements in the current row

**C** The current column in the printout

**J** The candidate prime number

**K** The index into the prime array for the last computed prime.

**JPRIME** Whether the candidate number (J) is “possibly prime” (false means definitely not prime).

**ORD** Leave as is, just lowercase. (TODO: come back to it?)

**SQUARE** the next possible prime square (TODO: come back to it?)

**N** Leave as is, just lowercase (TODO: come back to it?)

**MULT** An array of multiples

Next, use “Code -> Reformat Code” to properly indent and space out everything.

Now, use the “Move declaration closer to usages” intention on all the variables that are declared near the top without an initial value. (page number, page offset, row offset, candidate prime, current column, etc)

Test (TODO: and commit?)

## Step 2: Basic Method separation

Looking at the code, we can see that it splits in broadly two parts:

- The lines up to the end of the first while loop, that seem to do with computing the prime numbers.
- The lines after the first while loop, starting with the page number initialization, which seem to have to do with printing the numbers.

Select each while loop in turn, and use extract method on them. One method should be called `generatePrimes` while the other should be called `printNumbers`.

If there are any variables that are not being used right now (grayed out), use the “Remove variable” intention to eliminate them.

Run your tests to make sure they still run.

You should now have been left with about a dozen variables defined near the top of the main method, then passed into the two extracted methods. Let’s think about these variables. They fall broadly in two categories:

- Some variables are clearly “global constants”, settings that the user might want to adjust, namely: the number of primes to print, the number of rows per page and the number of columns per page.
- Others are used by the “generate primes” function in order for it to do its work. These are: `ordmax`, the multiples array, the candidate prime, the index into the prime array, `ord`, and the square. We really would like all of those to be local variables within the “generate primes” function, but they did not end up there.

In order to get this second group of variables into the function, there are a couple of ways but the simplest is to just undo our last step:

- Use the “Inline” refactoring to inline and remove the generate primes function. We will create it again in a moment.

- Use the “Move declaration closer to usage” intention, or simply move line-by-line, to move all the declarations of variables that we want to go as part of the “generate primes” function right next to the corresponding while loop.
- Select the while loop, along with all those variable declarations, including the primes array declaration and the primes[1]=2 part, and perform the “extract Method” refactoring. If you do this correctly, you should see only one parameter on this new method, namely the number of primes. And the method should be set to return the primes array.

Test (TODO: and commit?)

### Step 3: Extracting Classes

The two methods we have are a good start: They each handle a specific and disjoint set of variables. But they are still quite large. It also feels like they are dealing with two completely separate tasks: One is concerned with generating a list of prime numbers, the other is concerned with simply printing out numbers.

This suggests that the two methods should really belong in different classes, and perhaps should each form the basis for those new classes. We will now create these classes.

- Select the “generate primes” function call and perform the “extract Method object” refactoring. Make sure to ask it to move the method as well when asked. Call the new class PrimeGenerator.
- This was likely created as an inner class to the PrimePrinter class, but we would ideally like it to be its own class. Select the class and perform a “Move” refactoring to move it to the upper level.
- Now take a look at the newly created object. It has a generatePrimes method that is really no longer needed as its own method. We would rather keep the invoke method, maybe rename it later. So select the ‘generateNumbers function and perform the “inline” refactoring and remove the method.
- Now rename the invoke method to generatePrimes.

Run the test to make sure it still works.

Of course we’re far from done with that class: It still consists of one large method, we’ll need to do something about that.

Now it is time to consider the number-printing function. Take a moment to look at its parameters:

- The rows per page and columns per page variable 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 number of primes parameters feel more elements you should be providing to the number-printer when you call its invoke or print method.

We want to achieve this separation. For this reason, we will use a different refactoring to form the new class. Here are the steps for this:

- Select the print-numbers method call, and do the “extract Parameter object” refactoring. Select only the two rows and columns parameters for it, and put NumberPrinter as the class name (make sure you set it to Create New Class). This should turn the print-numbers function into a three parameter function, one of those parameters being a NumberPrinter instance.
- Perform the “Convert to instance method” refactoring, to turn the print-numbers method into an instance method of the NumberPrinter instance.
- Looking at the NumberPrinter class now, we see that IntelliJ has “encapsulated” the two fields behind getter methods. We don’t really need that, so select each method and perform an inline refactoring to remove them.

Now our main method should look particularly simple: Initialize a few parameters, create a prime number generator and call it, and create a number printer and call it with the primes. Let’s do some refactoring to clean it up though:

- Select the new NumberPrinter(...) call (without the printNumbers part), and perform extract local variable from it to create a numberPrinter local variable.
- Select the new PrimeGenerator(...) call (without the generatePrimes part), and perform extract local variable from it to create a primeGenerator local variable.
- Select the primes array and choose the inline refactoring to inline construction into its use.
- Looking at it here, let’s rename the printNumbers method to just print and the generatePrimes method to just generate. Remember to use the Rename refactoring for this!

Run your tests.

## **Step 4: Working on the NumberPrinter class**

Let’s see if we can do some cleanup of the NumberPrinter class:

- The parameter to the print method is called primes, but it clearly can be any array of numbers. So perform a Rename refactoring to change the parameter to numbers.

- The other parameter is the number of numbers to print. We don't really need that any more technically, as we can simply use the length of the numbers array instead. To do this in a stepwise fashion, do the following:
  - Find the first use of the number-of-primes variable, and use “Extract Variable” to create a local variable equal to it, and named numberOfNumbers. Make sure to replace all 3 occurrences.
  - Change the numberOfNumbers = ... assignment to instead be numberOfNumbers = numbers.length (The numbers in the array actually start at index 1) Run your tests to make sure they still work.
  - Now the parameter in the print method should be grayed out as it is not being used. Use the “Safe Delete” intention on it, and run your tests again.
- Let's continue with refactoring the print method into smaller parts. We start with the double for loop: Perform the Extract Method refactoring to it, to obtain a method printNumbersOnPage.
- It seems that the first group of System.out calls prints a header, so go ahead and extract the method printHeader from it.
- The remaining part of the while loops seems to update the counters as we advance through each page. It also seems that the methods we use all need those values. So we should elevate them to fields so that they can be more easily shared. Perform an “extract field” refactoring on the page number, page offset and number of numbers variables.
- Extract a method from the last three lines of the while loop, call it moveToNextPage.
- Now that we are using fields, the page number and number of number parameters in the printHeader method are no longer needed. Perform an “Inline” refactoring to them, and remove the superfluous this. parts that are added. Run your tests.
- Do the same for the printNumbersOnPage method, leaving just the numbers parameter to it.
- It feels that numbers should also be set to a field. Extract the field from the first occurrence of numbers, in the initialization of numberOfNumbers. Make sure to tell it to replace all occurrences.
- Run your tests and they should now fail. That's because IntelliJ did not actually add a statement to initialize this.numbers to equal the parameter numbers. Do so in the print method, and check that your tests are back in order.
- The numbers parameter in printNumbersOnPage should now be redundant. Perform the “Safe delete” intention on it.
- The first four lines of the print method are all about initializing fields. Extract them to a method initialize.

- The test in the while loop is determining if there are still more numbers to print. Extract it into a method `needToPrintMore` (keep the original signature when asked). This make it more clear what that test does.
- Let's shift our attention to the `printNumbersOnPage` method. Look at the conditional at the innermost level of the loop. Let's extract it to a method `printNumberAt`.
- Notice that this new method takes two parameters but really only uses one, namely their combination as "row offset plus column offset times rows per page". Extract parameter from that expression, and tell it to replace both occurrences. It should also remove the other two parameters in this case.
- Let's look at the row-offset variable of the outer loop. It seems to be initialized as page offset, then stop at a boundary similarly depending on page offset. It is only used in the index computation. Change it so that it instead starts at 0 and ends at rows-per-page minus 1, and change the index computation to include an additional page-offset. Run your tests to make sure they still pass. Maybe also perform a Rename refactoring to now call the variable `row`.
- Looking at the stopping tests in our for loops, we are more used to seeing them with a less than comparison, rather with a "less than or equal to the number minus 1" comparison. So fix those up, and run our tests to make sure they still pass.

## Step 5: Reducing the number of fields

It seems there are still too many fields in the `NumberPrinter` class. A primary example is the page offset. It should be simple enough to compute the page offset from the page number, and it is not really used in too many places. We would like to replace it with a computation. In order to achieve this, we will do the following:

- Use the "Encapsulate Fields" refactoring to encapsulate the get access for page offset. This will replace all accesses to this page offset by a call to `getPageOffset()`. Run your tests to make sure we didn't break anything yet.
- Now, we can edit the `getPageOffset` method to instead compute the offset from the page number and return it. The computation should be "page number minus 1 then times rows per page and times columns per page, then add one to the result." Run our tests to make sure this change did not break anything.
- Now, the page offset field should appear grayed out in its declaration point. Go ahead and do the "Safe delete" intention and run the tests again.

Next we have the number of numbers field. It's not used much, and we can just compute it from the numbers array instead. So let's do that:

- Use the "Encapsulate Fields" refactoring again, this time to encapsulate the getters of the number of numbers field.

- Replace the body of the new getter to instead return the numbers array length minus one. Run the tests to make sure they still run.
- Notice that the `numberOfNumbers` field appears grayed out now, and perform the “Safe delete” intention. Make sure the tests still run.

## Step 6: Parameterizing the title

The printing of the numbers includes some header information. The first part of that information is the title, the other is the page numbers. We should probably make the title into a parameter that our creators provide, as we don’t know what kinds of numbers they would want us to print. We’ll keep the page number logic as part of our work.

This all will happen in the `printHeader` method, which currently is a series of `System.out.print` calls. Our first task would be to bring them together.

- Step-by-step merge each of the first two calls into the next one, by prepending its string to the front of the argument. For example after the first step the first two line should have become one call, with argument `"The First " + Integer.toString(getNumberOfNumbers())`. Run your tests after each step.
- Do the same to bring the last two calls together.
- Eliminate the `Integer.toString` parts (leaving their arguments intact, letting the plus operator worry about adding strings to integers).
- The `" --- Page "` part belongs to the TODO

TODO