Activity 2-5a Refactoring: The `NumberPrinter` Class

This activity continues the refactoring of primeGenerator that you began in Activity 2-4.

Step 4: Refactor the `print` method

1. The first thing to clean up are the parameters for the print method: int numPrimes and int[] primes.

   a. Although the second parameters is called primes, in theory it could be any array of numbers. Perform a **Rename** refactoring to change the name of primes to numbers.

   b. The parameter numPrimes indicates how many numbers to print. We can just use the length of the numbers array instead, which makes the numPrimes parameter unnecessary. Fix this in a stepwise fashion as follows:

      - Find the first use of the numPrimes variable.
      - Use **Extract Variable** refactoring to create a local variable called numberOfNumbers. When asked, be sure to replace all 3 occurences. Also, renaming the variable will be the last thing you do.
      - Change the numberOfNumbers = ... assignment to instead be

        ```
        numberOfNumbers = numbers.length - 1
        ```

        (Note: the numbers in the array actually start at index 1.)
      - Run the tests to make sure they still work.

   c. The numPrimes parameter should now be grayed out as it is no longer being used. Use **Safe Delete** to remove it.

      - Select the parameter.
      - Use <alt>–ENTER to show the available intentions and select **Safe Delete**.
      - Run tests again.

2. The next thing to do is simplify the while loop by extracting various methods.

   a. The first five lines of the while loop are System.out calls. Together these lines print the header for a page. Select these lines and extract them into a method called printHeader.

   b. Next inside the while loop is the double for loop that is responsible for actually printing the numbers on the page. Extract a method called printNumbersOnPage from the double for loop.

   c. The last three lines of the while loop update the pageNumber and pageOffset counters as we advance through each page. These three lines also need to be turned into a method, but you notice that, pageNumber is being used by the printHeader method, and pageOffset is being used by the printNumbersOnPage method. Also, the variable numberOfNumbers is being used by both those methods. Before turning the last three lines into a method, pageNumber, pageOffset, and numberOfNumbers need to beneed to be elevated from local variables into class fields.

- Select any one of the pageNumber references and refactor to **Extract Field**. Keep the same name for the name of the field.
- Repeat the process to extract pageOffset and numberOfNumbers into fields as well.

d. Now extract a method called moveToNextPage from the last three lines of the while loop.

3. Clean up the parameters of the newly created methods.

   a. Because pageNumber and numberOfNumbers are now fields, they no longer nee to be passed into the printHeader method.
      - Go down to where the printHeader method is being defined.
      - Select pageNumber in the parameter list.
      - Use refactoring to **Inline** the parameter.
      - Repeat this process to inline the numberOfNumbers parameter.
      - Remove the superfluous this that was added to the pageNumber and numberOfNumbers references in printHeader.
      - Run tests.

   b. Repeat the steps above to inline the pageOffset and numberOfNumbers parameters used by the printNumbersOnPage method. If a line is created initializing pageOffset as a local variable, delete this line.

   c. numbers is now the only parameter that remains for printNumbersOfPage, an it makes sense to also extract numbers into a field.
      - Go back to the print method.
      - Select numbers in the line initializing numberOfNumbers, and extract it as a field. Be sure to check the box to replace all occurences.
      - Now if you run your tests, they should now fail, because IntelliJ did not actually add a statement to initialize this.numbers to equal the parameter numbers. Add the line this.numbers = numbers to the top of the print method.
      - Run tests to verify they are again passing.
      - Back in the printNumbersOnPage method, use **Safe Delete** to delete numbers from the parameter list, and remove the superfluous self. in the next-to-last line of the method.

4. The first four lines of the print method are now all about initializing fields. Extract these lines into a method called initialize.

5. The boolean expression in the while loop is determining if there are still more numbers to print. Select this expresssion an extract it into a metho called needToPrintMore. Keep the original signature when asked, and *do not* replace other occurrences when asked.

6. The last thing to do is to clean up the printNumbersOnPage method.

   a. Down to the printNumbersOnPage method, select the conditional inside the nested for loop and extract it as a method called printNumberAt.

b. To to the newly created printNumbersAt method. **NOT SURE WHAT TO DO HERE**

- 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 occurences. It should also remove the other two parameters in this case.

c. 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.

d. 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

The NumberPrinter class has quite a few fields. Some of these, like pageOffset, are not used in too many places and could be replaced by a simple calculation, e.g., calculating pageOffset from pageNumber.

1. Encapsulate pageOffset so that it is only accessible through an accessor method.

   a. Select pageOffset and refactor to **Encapsulate Fields**.
   b. Unckeck *Set access*. (The goal is to eliminate the need for this field, so it makes no sense to be creating a setter.)
   c. Refactor to replace all accesses to this field with a call to getPageOffset().
   d. Run tests to make sure nothing was broken.

2. Change the getPageOffset method so that it computes pageOffset from pageNumber and return it.

   a. The computation should be "page number minus 1 then times rows per page and times columns per page, then add one to the result."
   b. Run our tests to make sure this change did not break anything.
   c. Up at the top of the class, the pageOffset field should now be grayed out where it is being declared. Use **Safe delete** to remove the field.
   d. Run the tests again.

3. The numberOfNumbers field can also be removed with a bit of refactoring. It is not used much, an it can be computed easily from the numbers array.

a. Use refactoring on numberOfNumbers to encapsulate the field. Again, you only need to create a getter for this field, not a setter.
b. Replace the body of the new getter to instead return the length of the numbers array minus one.
c. Run the tests to make sure nothing is broken.
d. Remove the numberOfNumbers field from the class using **Safe Delete**.
e. Run tests.

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 with the second statement, not the first, so move it over and make sure your tests still pass.
- You should now have two System.out... statements, the first one setting the document title, the other setting the page number. We now want to turn the document title into a parameter. Select it and perform "Extract Parameter", name the parameter title.
- Going up to the print method, the title shows up there instead. Do another Extract Parameter to lift it to a parameter of the print method. Your tests should now be failing. Go to the PrimePrinter method and change the call to numberPrinter.getNumberOfNumbers into a reference to the number of primes method instead.
- Back in our printHeader method, put the two String.out statements into one, then go anywhere in the string and use the Replace "+" with String.format intention.

Step 7: Make the main loop clearer

There is something bothering us about the current structure of the main loop: It is supposed to be printing the next page every time, yet somehow its current structure doesn't allow for that. Part of the problem is that the page number is currently a field value, and getting updated in mysterious intervals: It is initialized in the initialize method, though nothing about the name of that method suggests that, then is updated at the end of the while loop, which feels a bit backwards. Ideally our loop, and print function, should say:

```
while there is a next page:
    print the next page
```

Even better, we should be able to simply say:

```
for page in pages:
    print page
```

Or in Java syntax:

```
for (int page : getPages())
    printPage(page)
```

In order to achieve this, we need to have an iterator. But before that, we need to have the page number as a parameter to the methods that form our for loop. The page number is used in a number of places:

- printHeader uses it to print the page number on the header.
- printNumbersOnPage uses it in its getOffset calculation.
- moveToNextPage actually increments it, which complicates matters considerably.

Let's work through this refactoring:

- We start with printHeader. Find the use of pageNumber in printHeader and perform Extract Parameter on it. Check that your tests still pass.
- Then, inline the moveToNextPage method and remove it. It won't really be doing much after we move around the page increment, so we'll just find a better place for the System.out call later. Do the same for the initialize method.
- Then look at the getPageOffset method, and perform "Extract Parameter" on the pageNumber variable there. Then move to the printNumbersOnPage method and perform "Extract Parameter" on the pageNumber variable from there.
- Finally, the needToPrintMore method also uses it via getPageOffset, so perform an Extract Parameter from there as well.
- Now we hopefully have isolated all the changes of the pageNumber field to the print method. It is set to 1 at the beginning of the print method, then incremented later on. You can confirm that the field is not used elsewhere by moving your cursor over the field declaration and using the "Navigate -> Declaration menu item". It should show you all the usages.

- Now with the cursor on the field declaration, choose the "Convert to local" intention. Then run our tests again to make sure everything works fine.

Step 8: Extracting a Page class

Thinking through the problem more, it almost feels like we need a separate class to capture the idea of the individual *pages*. Then that class can incorporate the logic about computing indices and knowing when it's done, for example. Perhaps we can call this new class a Page. Let's think through what it would need to know:

- It needs to know its number, currently stored in pageNumber.
- It needs to know the row/column dimensions.
- It needs to know the actual numbers array to be able to index into it.

So this class will kind of end up knowing almost all the same stuff as the pretty-printer (except for the title for example). But it does not concern itself with headers and footers for example, or where to output the values. And we might later consider other ways to paginate the page (e.g. numbers going row first). Let's give this a go:

- First, turn the pageNumber local variable back into a field (extract field). We are about to do an "extract delegate" refactoring, which works best with fields.
- Now perform the "extract delegate" refactoring, which allows you to pull apart fields and methods of one class to another. Name the new class Page, and include in it all the members except for print, printHeader and printNumberOnPage. Make sure to select the "generate accessors" box.
- Try to run your tests now, and they should fail. It looks like the problem is that rowsPerPage and columnsPerPage are marked as final, which is correct since they should really only be set once in the constructor. But that's not how they are set. So let's see if we can fix that:
  - Back in the NumberPrinter, notice how the page field is initialized at its declaration. Use the "move initializer to constructor" refactoring to bring that into the initializer.
  - Back in Page, go over the rowsPerPage field and use the "add constructor parameters" refactoring to select both fields and add them to the constructor as parameters.
  - Back in the NumberPrinter constructor, eliminate the two this.page.set... lines.
  - Back in Page, use the "Safe delete" intention to remove the various grayed out methods.
  - Run your tests now and make sure they run.
- Now let's do some more cleanup. There are some methods back in NumberPrinter that are not being used. Go ahead and use "Safe delete" on them as well.

- Now we need to do some cleanup. A number of methods in NumberPrinter are now grayed out, go ahead and use the "Safe delete" intention on them. Make sure your tests still run.
- There is a page.setPageNumber(1) call that really should not be needed, as that should be part of the initialization of the page class. Delete that call and instead initialize the pageNumber field in the Page class to 1.
- In the Page class, the getPageOffset method no longer needs the pageNumber parameter. Perform the Change Signature refactoring to eliminate its parameter. Run your tests to make sure they are OK.
- Next "Safe delete" the unnecessary parameter in needToPrintMore. While we are at it, perform a renaming of it to be simply called hasNext.
- Now let's shift our focus back to the NumberPrinter. Note the last line in the print while loop, which increments the page number by 1. This really should be a method of page. Select the whole expression and perform an "Extract method" refactoring to it, to a new method named nextPage. Then perform a "Move" refactoring to move it to the Page class. Now go to the body of this new method in the Page class, and perform "Inline" refactorings, selecting "this only and keep the method".
- Now the setPageNumber method is probably grayed out. You can perform "Safe delete" on it.
- Let's clean the class up a bit. There are four fields declared, move their declarations near the top, before all the methods. You can use the "Code -> Move Statement Up/Down" shortcuts.
- Similarly, move the constructor to be right below them.
- There are a number of get... methods. Move them all close to each other below the constructor.

Now let's look at the printNumberAt method. It feels as though that method is not quite doing what this class should be doing. We want this class to be about computing, for example to compute the index. But the actual printing would be left up to the NumberPrinter class. So our work needs to start from the print method in the NumberPrinter class.

- Look at the index computation in the argument to the printNumberAt call inside the inner for loop the printNumbersOnPage method. select that whole argument and extract a method for it: getIndexFor. Then move that to the page method. Then inline the getRowsPerPage call in it.
- Back in the printNumbersOnPage method, inline the call to printNumberAt (and remove the method as this is its only occurence).
- At this point the system created an index local variable. Go ahead and inline it to eliminate it.

- Now the test inside the if should be its own method, so extract it to a method hasEntry then move that method to the Page class.
- The second argument to the String.out.printf call should also be its own method, called getEntryAt. This is a bit tricky: Select it and start the "Extract Method" refactoring, and you will see that the "Fold parameters" checkbox is checked. Go ahead and uncheck it, then refactor it and move it to the Page class.
- As a final cleanup, the pageNumber parameter is not needed, so perform a "Safe delete" on it.

Let's return to the main print method. Remember that our goal was to simplify that loop a bit. Let's start by putting together the three methods in the while loop: The header printing, the page printing and the footer printing. Call the new method printPage.

Now the while loop looks a lot simpler! It still feels a bit off though, we should be going to the "next page" first. So move the page.nextPage() line up a step. This of course will fail our tests. We will need to adjust pageNumber in a number of ways:

- pageNumber should start 0 instead of 1.
- hasNext should instead use getNextPageOffset, a new method that is like getPageOffset but uses pageNumber instead of pageNumber – 1.

After you have made those changes and created the new method, your tests should again pass.

One final cleanup before calling it a day: The second parameter to printHeader can be inlined. Do that. And the System.out.println("\f"); line should really be a method called printFooter, so extract that, then rearrange the methods so that they follow the stepdown rule.

This activity continues in refactoring activity 3[1].

[1] activity2-5bRefactoringPrimesGeneratorPart3.html