

## 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 parameter 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 occurrences. 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 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 need 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 `printNumbersOnPage`, and 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 occurrences.
  - 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 expression and extract it into a method 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. Go to the newly created `printNumbersAt` method. Although the method takes two parameters, in both places where the parameters are used in the method, they are used in the same expression: `rowOffset + col * rowsPerPage`.

Instead of passing in two separate parameters, refactor and extract a parameter from the expression `rowOffset + col * rowsPerPage`. Check the box to replace both occurrences. The refactoring should clean up the original two parameters automatically.

- c. Go back to the `printNumbersOnPage` method. `rowOffset` is being used as the loop variable for the outer loop. Both the initial value for `rowOffset` and the stopping condition are based on the value of `getPageOffset()`, which just makes the loop harder to understand. Clean this up as follows:
  - Use refactoring to rename `rowOffset` to `row`.
  - In the call to `printNumberAt`, change `row` to `row + getPageOffset()`.
  - Change the outer loop so that `row` begins at 0 and ends at `rowsPerPage - 1`.
- Run the tests to make sure they still pass.
- d. Lastly, fix the stopping condition for each for loop to be a “less-than some value” comparison rather than a “less-than-or-equal-to some value minus one” comparison.

## 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. Uncheck *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 the page offset from `pageNumber` and returns the result.
  - a. The computation should be:  
$$(\text{pageNumber} - 1) * \text{rowsPerPage} * \text{colsPerPage} + 1$$
  - b. Run our tests to make sure this change did not break anything.
  - c. Now that the `pageOffset` field is no longer being used in a calculation, you can remove any occurrences where `pageOffset` was being assigned a value.

- At the top of the class where `pageOffset` is grayed out, select the field, refactor and use **Safe delete** to remove the field.
  - Safe delete should report about the getter you just created. *You should NOT delete this getter.*
  - When told about a usage that is not safe to delete, view the usage.
  - The bottom panel should show that `pageOffset` is being used in the `moveToNextPage` metho. Double clicking on the usage that is show will take you to that spot in the code.
  - Manually delete the line assigning a value to `pageOffset`.
  - Back in the bottom pane, click “Rerun Safe Delete”. This time, no `pageOffset` usages should be found and the field should be deleted.
- d. Run the tests.
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

Printing the numbers includes printing the header information, title and page number, for each page. Ideally, the title should be a parameter that the method caller can provide – there is no way of knowing what *kinds* of numbers the caller will ask the method. The logic for calculating the page number can remain unchanged.

1. Edit the `printHeader` method so that contains two calls: `System.out.print` to output the title and `System.out.println` to output the page number.
  - a. Use the “+” operator to concatenate the different string arguments together.
  - b. Eliminate the `Integer.toString` calls but keep their arguments; the plus operator can deal with adding strings and integers.

The new version of `printHeaer` should look like the following:

```
private void printHeader() {
    System.out.print("The_First_" + getNumberOfNumbers() + "_Prime_Numbers_");
    System.out.println("—_Page_" + pageNumber + "\n");
}
```

2. Select everything that is being passed to the first `Sytem.out...` call. Use refactoring to extract this into a parameter with the name `title`.

3. Go up to the print method. Repeat the process of extracting  
`"The_First_" + getNumberOfNumbers() + "_Prime_Numbers_"`  
into a parameter called `title`.
4. Run your tests, and you will discover that they are failing, which means something is now broken. Go to the `PrimePrinter` method in `PrimePrinter.java` and change the `numberPrinter.getNumberOfNumbers` call into a reference to the `numPrimes` field. This should fix the broken test.
5. Go back to `NumberPrinter.java` to the `printHeader` method.
  - a. Add `title` to the second `Sytem.out...` call and delete first `Sytem.out...` call, which is no longer needed.
  - b. Click anywhere in the argument to the `Sytem.out...` call and check available intentions. Choose **Replace "+" with String.format**.

## Step 7: Clarify the while loop in the print method

Back in the print method, consider the current structure of the while loop. This loop should be printing the next page every time it iterates, but its current structure does not allow for that.

Part of the problem is that the page number is represented by a field, and it is mysterious how it is getting updated: - It gets an initial value in the `initialize` method, but nothing in the name of that method tells you that explicitly suggests. - Then, the page number is being upated in the `moveToNextPage` method at the end of the loop, which feels a bit backward. is updated at the end of the while loop, which feels a bit backwards. Ideally, the loop should have the following structure:

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

or even better:

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

In Java syntax, this would look like:

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

To achieve this, you need an iterator. But before adding an iterator, you need to refactor the methods that are being called in the while loop to take the page number as an argument. All three method calls use the `pageNumber` field:

- `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.

The steps to accomplish this refactoring are below:

1. Go to the `printHeader` method and find the use of `pageNumber` in this method. Use refactoring to extract `pageNumber` into a parameter, and check that all tests still pass.
2. Go to the `moveToNextPage` method. Use refactoring to inline and remove the method. This method will not be doing much once the page increment has been moved around; you will find a better place for the `System.out...` call later.
3. Repeat the inline and remove refactor on the `initialize` method.
4. Go to the `getPageOffset` method. Find the use of `pageNumber` and extract it as a parameter.
5. Go to the `printNumbersOnPage` method and repeat the process of extracting `pageNumber` as a parameter.
6. Finally, go to the `needToPrintMore` method. This method uses `pageNumber` in its call to `getPageOffset`; again extract `pageNumber` as a parameter.
7. All changes to the `pageNumber` field should now be isolated to the `print` method. At the beginning of method, `pageNumber` is set to 1, and `pageNumber` is being incremented at a natural spot at the end of the `while` loop. You can confirm that the `pageNumber` field is not used elsewhere by moving your cursor over the field declaration and using the *Navigate -> Declaration* menu item. This action should show you all the places `pageNumber` is being used..
8. With the cursor on the `pageNumber` field declaration, choose the **Convert to local** intention.
9. Run your tests again to make sure everything is still working.

## Step 8: Extracting a Page class

Thinking through the problem more, it almost feels like there needs to be a separate class to capture the idea of individual *pages*. Such a class could incorporate the logic about computing indices and knowing when it is done, for example. Let's think through what this new `Page` class 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.

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. Later we might consider other ways to paginate the page (e.g. numbers going row first). The steps below give this refactoring a try.

1. Turn the `pageNumber` local variable back into a field (extract field). This prepares you to do an **Extract Delegate** refactoring, which works best with fields.
2. With the cursor anywhere in the `NumberPrinter` method, refactor to **Extract Delegate**. This refactoring will allow you to pull fields and methods of one class to create another class.
  - a. Name the new class `Page`.
  - b. Include all members in the `Page` class except for `print`, `printHeader` and `printNumberOnPage`. Make sure to select the “generate accessors” box.
3. If you run your tests now, they should fail, complaining that `rowsPerPage` and `columnsPerPage` are marked as `final`. This is actually correct, since these fields should really only be set once in the constructor. However, that is not how these fields are currently being set.
  - a. Go to `NumberPrinter.java`. At the top of the class you will find that the `page` field is being initialized as part of its declaration. Click anywhere in that line then view available intentions. Select **Move initializer to constructor** to move the initialization of `page` into the constructor.
  - b. Back in `Page.java`, select the `rowsPerPage` field and view available intentions.
    - Select the **Add constructor parameters** intention.
    - In the window that opens, select both `rowsPerPage` and `colsPerPage` to add as parameters.
    - Click “Refactor” to change the signature of the `Page` constructor.
  - c. The methods `setRowsPerPage`, `setColsPerPage`, and `getNumbers` are now no longer being used; remove them using **Safe Delete**.
  - d. Go to the `NumberPrinter` constructor, delete the two `this.page.set...` lines.
  - e. Run tests to verify that everything is once again working.
4. Finish up this part of the refactoring process with some cleaning up.
  - a. Use **Safe delete** to remove the methods in the `NumberPrinter` class that are no longer being used.
  - b. Inside the `print` method, there is a `page.setPageNumber(1)` call that really should be part of the initialization of the `Page` class. Delete that call in `print`, then go to the `Page` class and instead initialize the `pageNumber` to 1.
  - c. `pageNumber` is now a field for the `Page` class, which means the `getPageOffset` method no longer needs the `pageNumber` parameter.
    - Refactor and use **Change signature** to eliminate this parameter.

- Ignore the problem that is detected and continue.
  - Run tests!
- d. Use the **Safe delete** intention to remove `pageNumber` as a parameter for `theNeedToPrintMore` method.
- e. Go back to `NumberPrinter.java`. Note the last line of the while loop in `print` is using the `setPageNumber` method to increment the page number by 1. This really should be a method of the `Page` class.
- Select the whole expression `(page.setPageNumber(page.getPageNumber() + 1))` and use “Extract method” refactoring to turn it into a new method called `nextPage`.
  - Accept the suggested signature change and the change to the line of code.
  - Go to the `nextPage` method that was just created, and use refactoring to move the method into the `Page` class.
  - Go find `nextPage` in the `Page` class. In the body of this function, select `setPageNumber`. Use refactoring to inline **this only and keep the method**.
- f. Use the **Safe delete** intention to remove the `setPageNumber` method, which is no longer being used.
- g. To further clean up the `Page` class:
- Move the four field declarations so that they are together at the top of the class. (You can use the *Code -> Move Statement Up/Down* shortcuts.)
  - Move the constructor to be right below the field declarations.
  - Move the various `get...` methods so that they are together below the constructor.
- h. Go to the `printNumberAt` method. This method is not quite doing what it should be doing for this class. We want this class to be about computing, for example to compute the index, leaving the actual printing to the `NumberPrinter` class. To fix this you need to start at the `print` method in the `NumberPrinter` class.
- The `print` method calls `printNumbersOnPage`. Go to the `printNumbersOnPage` method and look at the index computation in the argument to the `printNumberAt` call inside the inner for loop. Select that whole argument and extract a method called `getIndexFor`.
  - Move the `getIndexFor` method to the `Page` class; inline the call to `getRowsPerPage` after you have moved the function.
  - Back in the `printNumbersOnPage` method, inline the call to `printNumberAt` (which will remove the method as this is its only occurrence).
  - At this point the system created a local variable (`i` or possibly `index`). Inline this variable to eliminate it.
  - The boolean expression inside the `if` should be its own method; extract it to a method called `hasEntry` and move that new 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” box is checked. Uncheck it, then refactor it and move it to the `Page` class.
  - As a final cleanup, **Safe delete** the `pageNumber` parameter in the `printNumbersOnPage` method.



5. Go back to the `print` method. Remember that the goal was to simplify the loop in this method. Start by extracting a new method out of the the function calls that make up the body of the while loop. Call the new method `printPage`. The while loop should look much simpler!
6. There are still some things off about the order things are happening in the new `printPage` method. We should be going to the “next page” first. Move the `page.nextPage()` line up a step so that it is the first line in the method body. This change, unfortunately, causes the tests to fail. To fix this requires `pageNumber` to be adjusted in a number of ways:
  - `pageNumber` should start 0 instead of 1; fix this in the field declaration at the top of the `Page` class.
  - `hasNext` should use a new method called `getNextPageOffset` instead of `getPageOffset`; this new method is like `getPageOffset` - 1<sup>1</sup>.

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

7. One final cleanup before calling it a day: The second parameter to `printHeader` can be inlined. Do that. And the `System.out.println("\n");` 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<sup>1</sup>.

---

<sup>1</sup>[activity2-5bRefactoringPrimesGeneratorPart3.html](#)