



CMSC 206: Data Structures
Assignment #1: Java Warmup
due on Gradescope by the beginning of class on February 1, 2018

Write the following programs in Java using Eclipse. You will upload your code to Gradescope following the instructions at the end of the assignment. Gradescope will run an *autograder* which will run tests against your programs and give you an indication of what grade you will get on the assignment. You may submit as many times as you like to get continued feedback from the autograder. *Note: The autograder is not currently working. I will aim to get it working by Friday evening 1/26.*

You will complete this assignment in Eclipse, following these instructions to get going:

1. Start up Eclipse.
2. When you are asked what workspace to open, choose to open the workspace you set up as part of Lab #1. (If you're on a different computer now, you may need to repeat the steps of Lab #1.) The workspace should be named *cs206* and is likely in your *Documents* or *My Documents* folder.
3. Create a new project by clicking the down arrow next to the "New" button  and choose *Java Project*.
4. Name the project *Warmup* and click *Finish*.
5. Create a new class with the  button.
6. Name the class *Factors*. (This is the name of the file for the first problem, on the next page.)
7. Click *Finish*.

1. Write a program named `Factors` (in *Factors.java*) that computes the prime factors of a number entered by the user. It should ask the user to enter a number and then print out all the prime factors (one per line) of that number. It should *not* print any number more than once, even if there is a duplicate factor (like how $12 = 2*2*3$). Here is an example:

```
Enter a number: 30
The prime factors are
2
3
5
```

Here is another example:

```
Enter a number: 28
The prime factors are
2
7
```

It is non-sensical to report prime factors for a number less than 2. If the user enters in a number less than 2, report an error. Here is an example of this kind of error:

```
Enter a number: -3
Invalid entry. Please enter a number >= 2.
```

Your program does not have to gracefully handle the possibility that the user will enter a non-number (e.g., letters).

As this will be graded (in part) automatically, your output format must exactly match these examples. (Future assignments will be more about individual methods instead of overall programs. You will be able to customize the input/output of those programs. For now, though, your program must behave exactly as specified here.)

You may adapt the code from the *Prime.java* file posted on the syllabus page, although a clever implementation will not need to use a primality check at all.

2. Write a program named `PrimeRange` (in *PrimeRange.java*) that asks the user for two numbers and prints out all the prime numbers between those two inputs, inclusive. (Here, *inclusive* means that if either number that the user enters is itself prime, then it should be included in the output.) You must use a method `private static boolean checkPrime(int n)` to do the primality checking. Here is its full description:

```
/** Determines whether or not a number is prime
 *
 * @param n The number to be checked
```

```
* @return True if and only if the number is prime
*/
private static boolean checkPrime(int n)
```

Here is an example of the program running:

```
Enter the first number: 14
Enter the second number: 29
The primes between 14 and 29 are
17
19
23
29
```

Like the previous program, your program must gracefully handle silly user input. Specifically: if the lower bound is less than 2 or if the upper bound (the second number) is less than the first, it is an error. Here are some examples of erroneous program runs (each stanza is a separate run of the program):

```
Enter the first number: -3
Invalid entry. The lower bound must be >= 2.
```

```
Enter the first number: 7
Enter the second number: 5
Invalid entry. The upper bound must be >= the lower bound.
```

3. Write a program named `MostPrimeFactor` (in *MostPrimeFactor.java*) that asks the user for a number and prints out which of its prime factors has the greatest multiplicity. The *multiplicity* of a prime factor is the exponent it must be raised to when using prime factors to reconstitute a number. For example, the number 12 has two prime factors: 2 (of multiplicity 2) and 3 (of multiplicity 1). In other words, $12 = 2^2 3^1$. The number 270 has 3 prime factors: 2 (of multiplicity 1), 3 (of multiplicity 3), and 5 (of multiplicity 1). In other words, $270 = 2^1 3^3 5^1$.

Here are some example runs of the program (with each stanza being a separate run):

```
Enter a number: 12
Prime factor 2 has the greatest multiplicity
```

```
Enter a number: 270
Prime factor 3 has the greatest multiplicity
```

```
Enter a number: 300
Prime factor 2 has the greatest multiplicity
```

```
Enter a number: 3
Prime factor 3 has the greatest multiplicity
```

```
Enter a number: 1
Invalid entry. Please enter a number >= 2.
```

If multiple prime factors have the same multiplicity (as is the case for $300 = 2^2 3^1 5^2$), report the least such prime factor. Also, following the previous programs, you should print out a helpful error message (as above) if the user enters in a number less than 2.

You may *not* use any aggregate data structure for this problem. An aggregate data structure is a data structure that holds many of the same items. In other words, no arrays or ArrayLists (or other data structure that can act like an array).

4. Make a new file *reflections.txt* (you can use the *New* wizard button at the left of Eclipse's toolbar) with answers to the following questions:
 1. How long did this assignment take you?
 2. What challenged you the most in this assignment?
 3. Whom did you work with on this assignment?
 4. What resources did you consult while doing this assignment?
 5. Do you have any feedback to offer about this assignment?
 6. Any other comments or questions?
5. Submit your work on Gradescope.
 1. Select the *Warmup* project within Eclipse.
 2. Right-click and choose to *Export*...
 3. The *Export* dialog box appears. Under *General*, choose *Archive File* and click *Next*.
 4. Make sure your *Warmup* project is selected in the list at the top left of the window and that *Save in zip format* and *Create directory structure for files* are selected toward the bottom.
 5. *Browse...* to find a good spot to save the file that will be exported. Name the file *Warmup.zip*.
 6. Click *Finish*. Eclipse will create a *Warmup.zip* file as directed.
 7. Log into Gradescope and submit your *Warmup.zip*.