Late Day Clarifications

Pseudocode
The syllabus says that you can use Late Days for late projects - however, the syllabus does not explicitly say how this works with pseudocode. The SLs and I talked it over, and we decided on the following policy:

- Late days cannot be spent on pseudocode. Turn it in on time, or you’ll just miss that part of the project grade.
- You must turn in your pseudocode using turnin; email will no longer be accepted.

(We were flexible about both of these points in the first two projects - but starting with Project 3, we’re going to enforce this.)

Java code
Starting with Project 3, if you email us your code (instead of using turnin), it will count as a Late Day, even if you do it before the deadline. Please use turnin. And remember: if you are having trouble connecting from home, it’s always possible to come to the 228 lab, and upload your file using a thumbdrive.

1 Introduction
In this project, you will write two programs. In the first program, you will read words from the command line. You will print out a duplicate of the command line several times - but each duplicate will be missing exactly one of the words. You will then do the same thing, but with the words from the command line in reverse order. To do this, you will implement a method which will print out all of the elements of a String array - except one.

In the second program, you will generate a list of prime numbers, and store them into an array; after you have generated all of them, you will print them out. To help you find the prime numbers, you will write two methods, which search through the integers to find the next prime number - and then return that number.
2 SkipArgs Overview

In program 1 (SkipArgs), you will take the command line arguments, and print them out several times. Each time that you print them out, you should print them out all on the same line (separated by a single space); however, each time that you print them out, you should skip exactly one of them - and replace it with a hyphen. Do this for each of the arguments, starting with the first.

After you have done this, create a new array, and copy the command line arguments into it. But when you do so, reverse their order. Now print the lines a second time, again skipping one of the arguments each time.

As part of this program, you must use a method; each time that you call it, it will print out one of the lines of the output (including the spaces between the words, and the newline at the end).

2.1 Example Output 1

Suppose that you ran your program with the command line arguments

    foo bar baz fred

Your program should produce exactly the following output:

    - bar baz fred
    foo - baz fred
    foo bar - fred
    foo bar baz -
    - baz bar foo
    fred - bar foo
    fred baz - foo
    fred baz bar -

2.2 Example Output 2

Suppose that you ran your program with the command line arguments

    asdf jkl

Your program should produce exactly the following output:

    - jkl
    asdf -
    - asdf
    jkl -

2.3 Example Output 3

If you run your program with no command line arguments, then it should print nothing.
2.4 Program Requirements

To receive most (but not all) of the points for this assignment, your program must:

- Print out each of the required lines of output
- Skip the proper element on each line
- Use a method to do the printing (main() should not include any calls to System.out.println() or System.out.print())
- Include a single space between each word (an extra space, either at the beginning or end of the line, is OK - but for full credit, you’ll have to remove it)
- Print a newline at the end of each line
- Make a duplicate of the array, with the values reversed, and use that to print out the second set of lines

To receive full credit, your program must additionally do the following:

- Do not print an extra leading or trailing space on each line; this means that you will print one fewer space than there are words
- Include a check in the printExceptOne() method (see below), which makes sure that the “skip” index is valid - and throw an IllegalArgumentException if not

2.5 About the Method

Your program must use a method to do all of the printing. It must fulfill the following requirements:

- It must be named printExceptOne.
- It must take exactly two parameters: an array of String, and an integer.
- It must print out exactly one line (including the newline character at the end). Do not have it print out multiple lines!
- It must skip the index given by the second parameter (it must print out a hyphen instead).

Since your program needs to print out many lines but the method can only print out a single one, you will need to write main() so that it will call this method multiple times.
3 PrimeFinder Overview

In program 2 (PrimeFinder), you will take two command line arguments: a start value, and a count. You will build an array of integers, of size ‘count’, and fill it with the primes that follow the start value.

Your program must generate all of the values, and store them into the array, before you print any of them out. When you print them out, print them out one per line.

3.1 Example Output 1

Suppose that you ran your program with the command line arguments

```
5 10
```

Your program should produce exactly the following output:

```
5
7
11
13
17
19
23
29
31
37
```

3.2 Example Output 2

Suppose that you ran your program with the command line arguments

```
16 3
```

Your program should produce exactly the following output:

```
17
19
23
```

3.3 Program Requirements

To receive most (but not all) of the points for this assignment, your program must:

- Convert the two arguments to integers. (For partial credit, it is not required that you check the length of the \texttt{args} array first. For full credit, that will be required.)
Allocating an array of integers of the proper size.

Fill the array using a loop. Each pass of the loop should call the `findNextPrime()` method (detailed below) to generate another prime number.

After the array is full, print out the contents of the array, one number per line.

**To receive full credit**, your program must additionally do the following:

- Check that the user gave the proper number of command line arguments. If the count is invalid, then print an error message and then terminate the program. Do **not** allow your program to crash with an `ArrayIndexOutOfBoundsException`!

  **NOTE:** It is OK if your program crashes with a `NumberFormatException`, if the user passes something other than an integer on the command line.

- Before creating the array or generating any primes, check to make sure that all of the following are true. (If any check fails, print an error message and terminate the program.)
  - The count must be $\geq 0$. (0 is not an error; simply print nothing.)
  - The count must be less than one million.
  - The starting value must be $\geq 2$.
  - The starting value must be less than one billion.

- `isPrime(int)` (see below) must immediately return `false` for all numbers which are even (except 2) or multiples of 3 (except 3).

That is, if the number is a multiple of 2 or 3, this method should return an answer without ever using any loop. Use a loop only for values that are **not** multiples of 2 and/or 3.

### 3.4 About the Methods

Your program **must** use **two** methods (in addition to `main()`). `main()` will call the first method; the first method will call the second.

The purpose of the first method is to scan through a list of integers to find the next prime number. It must fulfill the following requirements:

- It must be named `findNextPrime`.
- It must take exactly one parameter: an integer.
- It must return the first prime which is $\geq$ its parameter. That is, `findNextPrime(2)` must return 2; `findNextPrime(12)` must return 13; `findNextPrime(21)` must return 23.
• It must call \texttt{isPrime(int)} to determine whether or not each number is prime.

The purpose of the second method is to check to see if a single number is prime. It must fulfill the following requirements:

• It must be named \texttt{isPrime}.

• It must take exactly one parameter: an integer.

• It must return a \texttt{boolean}: \texttt{true} if the parameter was a prime number, and \texttt{false} if not.

4 Pseudocode

Download the latest pseudocode requirements from: \url{http://www.cs.arizona.edu/classes/cs127a/spring16/projects/}

Project 4 will be graded based on v1.2 of the requirements.

When you write the pseudocode for these programs, you don’t need to describe how you will get the inputs from the command line - you’ll do that in the Java code. You may simply assume that you have the right number of variables, which hold the inputs, when your pseudocode begins. (Make it clear what their names are.)

You should write two different pseudocode files, one for each program. Give them different names, and turn them both in.

For hints about writing pseudocode, check the Project 1 spec.

4.1 Pseudocode and Methods

The new version of the Pseudocode Requirements document shows how to declare a method in pseudocode - and also how to call it. Make sure you read the new information (it’s at the end of the document).

5 Java code

After you complete the pseudocode (and turn it in!), start working on translating it to Java. (For a list of steps for converting the pseudocode to Java, see the Project 1 spec.)

Your classes must have exactly the names given above: \texttt{SkipArgs} and \texttt{PrimeFinder}. Make sure that your Java files are named to match.

5.1 Header Comment

Every Java file that you submit must have a \textbf{Header Comment} - which is a simple comment, at the top of your file, which gives basic information about it. A header comment must include:
• The name of the Java class
• The name of this class ("CSc 127A Spring 16")
• The assignment name
• Your name
• Your Section Leader’s name (or letter)
• A quick description of the Java class.

Here’s my suggested header comment for Program 4:

```java
/*
 * class ScanString
 * CSc 127A Spring 16, Project 04
 *
 * Author: TODO: your name here
 * SL Name: TODO: the name of your SL here
 *
 * ---
 *
 * TODO: put a quick (2-3 sentence) description of the class here
 */
```

6 Turning in Your Programs

See the class website for information about how to upload your files to lectura, and how to use the ‘turnin’ utility.

6.1 Pseudocode

Your pseudocode file must be either TXT or PDF files (one per program), but you may name them whatever you want. You must turn them in using the assignment name

```bash
cs127a_s16_sX_proj04_pseudocode
```

(replace the X with your section letter). **REMEMBER:** The pseudocode is due two days before the Java code!

6.2 Java Code

You must turn in file(s) named exactly this:

```bash
SkipArgs.java
PrimeFinder.java
```

You must turn in the Java file(s) using the assignment name
cs127a_s16_sX_proj04

(replace the X with your section letter).