Homework 1 -- Warmup
CSPP 51036
Due: Monday Oct 11, before class

Submission instructions
 -Create one .java file for each problem:
  1. ReverseEcho.java
  2. Summation.java
  3. CharCount.java
  4. Uppercase.java
  5. Adder.java
  6. Reorder.java
  7. StudentSorter.java
  8. Matrix.java
  9. Date1.java
  10. Date2.java

 -tar into single archive <yourLastName.tar>
  For example, I would create my homework tar as: tar cvf siegel.tar *.java
  (assuming I had only 6 above-listed .java files in directory)
 -email .tar as attachment to grader

Do NOT:
 - submit .class files
 - submit ParserUtils.java or TextManipTools.java (if you choose to use)
 - submit test input files (e.g. for #6)
 - resubmit after submitting first time
 - miss deadline

Comment on error handing:
 I do not specify exacly how to handle every possible error
 condition. In general, I expect you to handle obvious user input
 mistakes but not necessarily every possiblity at this point. If you
 are unsure what qualifies as "obvious", please simply mimic the
 reference implementation to be safe.

reference implementation to be safe.

0. Read chapters 1-3 in Horstman

1. Write a command-line application that reverses the order of the
   input arguments. You do not have to preserve multiple white-spaces
   between tokens.

Specification:
   java ReverseEcho arg1 [arg2 [arg3 ...]]
   arg1: One or more tokens which will be echoed in reverse order

   example:
    prompt>> java ReverseEcho Hello My name is Andrew
    Andrew is name My Hello

   Error Handling
    Must elegantly handle the following situations:
     - no input

2. Write a command-line application that sums all values from some
   user-specified min to some user-specified max.

Specification:
   java Summation <minval> <maxval>
   minval: integer starting val
   maxval: integer ending val

   example:
    prompt>> java Summation 2 5
    Sum: 14

   Error Handling
    Must elegantly handle the following situations:
     - maxval is < minval
     - too few arguments
     - too many arguments

3. Using my TextManipTools.java class, write a command-line program
that counts all occurences of a user-specified character within
a user-specified file.

Specification:
java CharCount <char> <filename>
char: character to look for. Can be either case-sensitive or case
insensitive -- you decide how you want to implement.
filename: name of file to look in

example:
prompt>> java CharCount e homework1.txt
Fifteen occurences of either 'E' or 'e' found.

Error Handling
too many/few input values
single character not input

4. Using TextManipTools.java, write a command-line program that
operates on a user-specified file and converts to uppercase the
first character following any period.

Specification:
java Uppercase <filename>
filename: name of file to operate on

example:
prompt>> java Uppercase someFile.txt
<new file contents are printed to stdout>

Error Handling
Can assume input is valid

5. Write a scripting interface that accepts the following commands:

The program must run a continuous loop and not shut down after
each operation -- that is, continously asks for input until "exit"
is typed.
(hint: using ParserUtils.java will make much easier)

Specification:
Run as: java Adder
This throws up a prompt which accepts the following syntax:

```
PROMPT>> <number1> + <number2>
PROMPT>> <number1> - <number2>
PROMPT>> <number1> * <number2>
PROMPT>> <number1> / <number2>
PROMPT>> <number1> % <number2>

example:

PROMPT>> 3 + 6
  9
PROMPT>> 4 - 3
  1
```

Handle gracefully as many incorrect inputs as you can. For example:

```
ex. ADDER > 3 $ 6
                        error: No such operator '$'

ex. ADDER > 3 + 2 + 1
                        error: must supply only two numbers

ex. ADDER > a + 2
                        error: symbol 'a' not a valid real
```

6. Write a program that reorders an integer array arr in-place according
to a vector of indices perm. For example,

arr = {0,4, -1, 1000)

perm= {3,0,1,2}

after a call to reorder such as,
reorder(arr,perm)

arr upon return has the values {1000, 0, 4, -1}

That is, arr is reshuffled such that, after reorder is called,
arr_new[i] = arr_old[perm[i]]. Note that arr should be overwritten
in memory, so arr_new and arr_old are are the same storage.

For simplicity, your program should be testable from the command line as:

>> java Reorder 0 4 -1 1000 & 3 0 1 2

Note: You may _not_ allocate a separate memory buffer of size
arr. Imagine that memory is at a premium when you write your
algorithm. Also, your program must internally use arrays in the manner
specified above. You can not simply print the output to the screen on
the fly.

7. Write a program which sorts a formatted text input file (read using
TextManipTools.java). The input file has the following structure:

<Last name> <First Name> <Qtrs-in-House> <Qtrs-in-college>
<Last name> <First Name> <Qtrs-in-House> <Qtrs-in-college>
.
.
.

For example, the file might look like this:

Siegel Andrew 6 3
Jones Bob 6 1
.
.
.

and so on.

The sorting rules are as follows: First by quarters in House, then by
quarters in College, then randomized where ties occur. Output should
go to stdout, so the program is run as:

>> java StudentSorter <infile>

8. Write a program that computes the product of two matrices. Time this
routine using the appropriate java System methods and graph the execution time as a function of matrix size. Compare these
results with the same alogirthm written in C.

9. Using no date-specific Java libraries, write a program that computes
   the number of days between any two dates (inclusive). You must take
   into account leap years. The rule for computing a leap year is as
   follows: If the year is divisible by 4 it is a Leap Year ...
   Unless the year is divisible by 100, then it is _not_ a Leap Year
   ... Unless the year is divisible by 400, then it _is_ a Leap Year.

   During a Leap year, an extra day is added at the end of the month
   of Feb.

   ex. 1000 is not a Leap Year

2000 is a Leap year

Your program must be run as follows:
PROMPT>> java DaysBetween 11/11/2002 11/12/2002
1 day
(program ends)

Error checking. Be sure to handle at least the following:

- if there is too few or too many input parameters
- if the date strings are not in the proper format (simply echo usage statement);
- if date 1 is not earlier than date 2

10. // Goal: Practicing learning a new Java API
// When covered: Second week (you will have to spend some time understanding the API)
Study the java.util.Calendar API. Redo problem (9) using the appropriate
methods of this class to carry out the computation.