In this assignment, we will be writing the code for the Exam 1 dot product question:  
  
  
  
Given the following definition of a dot product, **write a complete JAVA program (in a single file)** that will match the interface / specification below. Will likely find that the provided JAVA documentation to be helpful.

A math equations on a white background

Description automatically generated

The program dp.java interface usage will be:

Usage: java dp <input file name 1> <input file name 2>

At commandline:

./java dp A B

Would print (assuming the data in the files were the as in the example above)

DP = 3

**IMPORTANT \_\_ Assumptions / requirements**: the data in the input data files are stored as **binary int data**, with a single int as the metadata for the file, describing the length of the array associated with that file. The data must also be read into appropriate arrays and then a loop should be utilized to do the computation. The computation itself must be done in a method with the following signature: **public** **static** **int** calc\_dp(**int**[] A, **int**[] B). You MUST match the interface behavior above.

The code should also so as much error checking as possible.

**PART 1:**

Make a program makeData.java with this interface:

USAGE: java makeData <number of items> <L> <U> <output file name>

It will create an output data file (that can be read by the dp.java program and specification for the data format above). It will produce random integers from UNIF[L,U], inclusive of each bound (like in prior assignments). The RNG will be seeded with the number 42. Make sure it has timing added.

**PART 2**

Complete dp.java – as per the instructions above, and you may start with the code that we used in class (given below).

Add timing information (use nano second timer) and appropriate outputs to the screen that would be a single row of data:

N T\_W T\_O DP

N: number of elements in the arrays

T\_W timing (in seconds N.XX scientific notation format)

T\_O timing of just the for loop for the DP calculation

DP the final result

Do plenty of error checking.

**PART 3**

Write a python script that will have this interface:

python gather\_data.py <N> <M> <step size> <L> <U> <results.txt>

The python script will act like the BASH / .BAT / .PS scripts from the prior assignment by repeatedly calling makeData and dp and gathering the data from each experiment starting with array length for the DP of N and ending at M (including M) with a multiplicative step size of step\_size.

The output to the screen from dp should go to the file passed above as results.txt.

After running it say for example like this:

python gather\_data.py 10 1000 10 -5 5 will.txt

If we cat will.txt, we’d expect to see (with example made-up times and dp answers):

N T\_W T\_O DP

10 1.32E-6 3.56E-5 -32

100 9.32E-5 6.56E-4 23

1000 4.32E-3 2.56E-2 64

(there needs to be a single header row and then the data for each experiment)

**PART 4**

Write a python script that will have this interface:

python plot\_data.py <results.txt> <outfile.png>

for example, if we ran:

python plot\_data.py will.txt my\_plot.png

The program would open the data in that file and create an appropriately formatted plot (in PNG format) that shows both the overall and work times on the Y axis versus the problem size (N) on the X axis.

Create a python requirements file so that I can replicate your environment on my own machine.

**SUBMISSION:**

Submit project with this structure. Make sure you err on the side of too much rather than too little.

./hw-06-your\_user\_id

./data

./code // including the plotting, etc.

./report

ZIP and submit in Moodle.

**Appendix:**

**import** java.io.DataInputStream;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.IOException;

**public** **class** dp {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**if**(args.length!=2) {

System.***out***.println("Usage: java dp <input file name 1> <input file name 2>");

System.*exit*(1);

}

DataInputStream FileA = **null**;

DataInputStream FileB = **null**;

**try** {

FileA = **new** DataInputStream(**new** FileInputStream(args[0]));

FileB = **new** DataInputStream(**new** FileInputStream(args[1]));

} **catch** (FileNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**int** numElementsA, numElementsB = 0;

**int** A[] = **null**; **int** B[] = **null**;

**try** {

numElementsA = FileA.readInt();

numElementsB = FileB.readInt();

**if**(numElementsA!=numElementsB) {

System.***out***.println("files are not the same length");

System.*exit*(1);

}

A = **new** **int**[numElementsA];

B = **new** **int**[numElementsB];

**for**(**int** i=0; i<numElementsA; i++) {

A[i] = FileA.readInt();

B[i] = FileB.readInt();

}

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

// int A[] = {1, 3, -5};

// int B[] = {4, -2, -1};

**int** dp = *calc\_dp*(A, B);

System.***out***.printf("DP = %d\n", dp);

}

**public** **static** **int** calc\_dp(**int**[] A, **int**[] B) {

// assuming that the dimensions match

**int** dp = 0; // start running sum off at 0

**for**(**int** i=0; i<A.length; i++) {

dp += A[i]\*B[i];

}

**return** dp;

}

}