CS 2, Spring 2018
Project #3.1 – Efficiency of Array vs. ArrayList
Due Feb 4

Project Description

Using the list operations fill, increment, and search, investigate whether arrays or ArrayLists are faster, or whether they are about the same for **int** and **float** values. This will also test times to generate both **int** and **float** random numbers as well as the time cost of automatic expansion of an ArrayList. Prior to running the program, predict for each comparison which will be faster and by how much (just a little faster or significantly faster).

Remember: **int** and **float** are used in simple arrays, **Integer** and **Float** are used in the ArrayList.

All results will be written to a single file named **`P3Output.txt`**
(Hint – open the file for append)

Background

Study these and write small practice programs as needed before starting the full project.

- data structures
  - one-dimensional array (not Java API array classes)
  - ArrayList, including basic methods: .get, .set, .add, and .clear
- Java classes
  - DecimalFormat for real numbers and for commas
  - Random to generate list values
- Wall clock vs CPU time
  - Details on clock time:

```
http://www.oracle.com/technetwork/articles/java/jf14-date-time-2125367.html
https://docs.oracle.com/javase/8/docs/api/java/time/package-summary.html
```

  - Details on CPU time:

```
https://docs.oracle.com/javase/8/docs/api/java/lang/Runtime.html
http://memorynotfound.com/calculating-elapsed-time-java/
https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#nanoTime--
```

Design & Program Structure

- The program has one class. Main calls methods for each of the operations (fill and increment – see below). The methods each do their own timing and output.

- Overall program design:

```
constants and instance data

constructor
fill method
increment method
main method
    create object of this class type to call methods
    call fill method
    call increment method
```

- Begin by creating a stubbed form of the program that compiles and runs, but does nothing beyond print messages from each method ("Now in method <blah>.").

Designing the Timing

Timing is only done on the operations, not setting up. For example, timing the **int** and **Integer** lists should use this approach, and similar for the **float** and **Float** lists:

declare an array of 100,000 **int**

declare an ArrayList of **Integer** with the initial capacity of 100,000
declare four variables, start & stop wall clock times and start & stop CPU times.

start the wall clock and CPU clock
for 0 to 9,999
    generate a random int for each array element
end for
stop the two clocks
calculate the time = stop – start
open the output file, write the details to a file, then close the file.

Operations to test

- Fill method. Does it take longer to fill an array or an ArrayList? Time the fill operation for both lists. Use random values. These values will not be used later, so the fact that the lists have different values does not matter.

- Increment method. Does it take longer to add one to each element in an array or an ArrayList? 1) Using the four existing structure (int array, int ArrayList, float array and float ArrayList), time how long it takes to add 1 to every element in the lists.

  Note: For ArrayLists you cannot directly increment a value; you must use `.get` to extract the value, add one to it, then use .set to place the updated value back in the list `(list.set(index, value))`.

Input / output

- This program has no input.
- Output goes to the file, you may also wish to display it on the screen for convenience. I will not accept screen caps of the output.

- The numeric values must be formatted and the times must be in the same scale. Read the documentation on wall clock and CPU time, as they report different time scales.

- Output format as below:

```
Run # x
Fill the list: Number of elements: nnnnn
    int array – wall clock          :  xxx.xxxx seconds
    int ArrayList – wall clock      :  xxx.xxxx seconds

    int array – CPU time            :  xxx.xxxx seconds
    int ArrayList – CPU time        :  xxx.xxxx seconds

    float array – wall clock        :  xxx.xxxx seconds
    float  ArrayList – wall clock   :  xxx.xxxx seconds

    float array – CPU time          :  xxx.xxxx seconds
    float ArrayList – CPU time      :  xxx.xxxx seconds


Increment elements in the list:
    int array – wall clock          :  xxx.xxxx seconds
    int ArrayList – wall clock      :  xxx.xxxx seconds

    int array – CPU time            :  xxx.xxxx seconds
    int ArrayList – CPU time        :  xxx.xxxx seconds

    float array – wall clock        :  xxx.xxxx seconds
    float  ArrayList – wall clock   :  xxx.xxxx seconds

    float array – CPU time          :  xxx.xxxx seconds
    float ArrayList – CPU time      :  xxx.xxxx seconds
```

As noted later in this assignment, you will be making 3 to 5 runs for each of these tests.

Note: if needed, take the decimal fraction out to no more than 8 positions.

Other specifications

- Work on one operation at a time; get it working and tested before moving on. The first operation will provide a model for the others. Also write up the results of each operation as you finish them (cover letter discussions).
- Random numbers. The lists will be filled with random values. Sample code, which requires importing the Random class in java.util:

```
Random randomGenerator;   // class level
   :
randomGenerator = new Random();   // in constructor
   :
// insert the upper bound, generates values 0 to upper bound - 1
someNumber = randomGenerator.nextInt( ? );    // each operation method
```

- Start by using N = 100,000 (stored in a constant) for the size of the lists. If you don't see any differences (i.e., if, under one of the lists and operations, array and ArrayList have about the same performance), place a loop around the operation to run it more times. For example, a *for* loop from 1 to 5 will make it run 500,000 times. What you need will depend on the speed of your computer.
- Use 10,000,000 (also a constant) as the upper bound for data values (elements in the lists).

- Format all numeric output using the `DecimalFormat` class (`import java.text`). The constructor takes a String pattern for each formatter: a pattern of "0.00" will give two decimal places to time output (seconds); a pattern of "0,000" will insert commas for large integer values such as the list size. This will take two different formatters. Example of creation and use of one formatter:

```
DecimalFormat realFormatter;   // class level
   :
realFormatter = new DecimalFormat("0.00");   // in constructor
:
… realFormatter.format(some value)   // as needed
```

- Clear the ArrayList using the .clear method at the beginning of each of the 3 to 5 test runs.

Project development

- Follow development guidelines, including starting with a stubbed program, working incrementally (compiling, running, and testing one component at a time), implementing simple specifications first, and checking the specifications frequently.

Discussions (cover letter)

- For each operation using both array and ArrayList (fill, increment), state which data structure (list) was faster, and whether the difference was significant.
- Your conclusions should be based on multiple runs (3 to 5 runs) to ensure that small differences (perhaps from different data) are smoothed out. For each of the multiple runs, clear out the lists so you won't be working with old data.
- Were the results as you expected, or were some surprising or counter-intuitive?
- List the topics you wrote small programs on before doing the full project. Was this development approach helpful?

Collaboration

- Work on this project alone, except for any office hours or email help needed.

Project submission

Ask before submission if you are uncertain about any specifications. Check documentation and style guidelines, and project submission policies.

Due date: Wed., Feb. 7
Late date: Sun., Feb. 11
Note that the "grace period" is only 4 days, not a week.

Deliverables

- cover letter with discussion questions given above
- source code file named **`P3Program.java`**
- output file named **`P3Output.txt`**
- any additional work (explained in the cover letter)

Projects: course policies

Same as before – if you don't remember, check the assignment specs for Eliza or Binary
Except when explicitly stated, all submitted work must be your own. Ask if you are uncertain about what constitutes plagiarism.