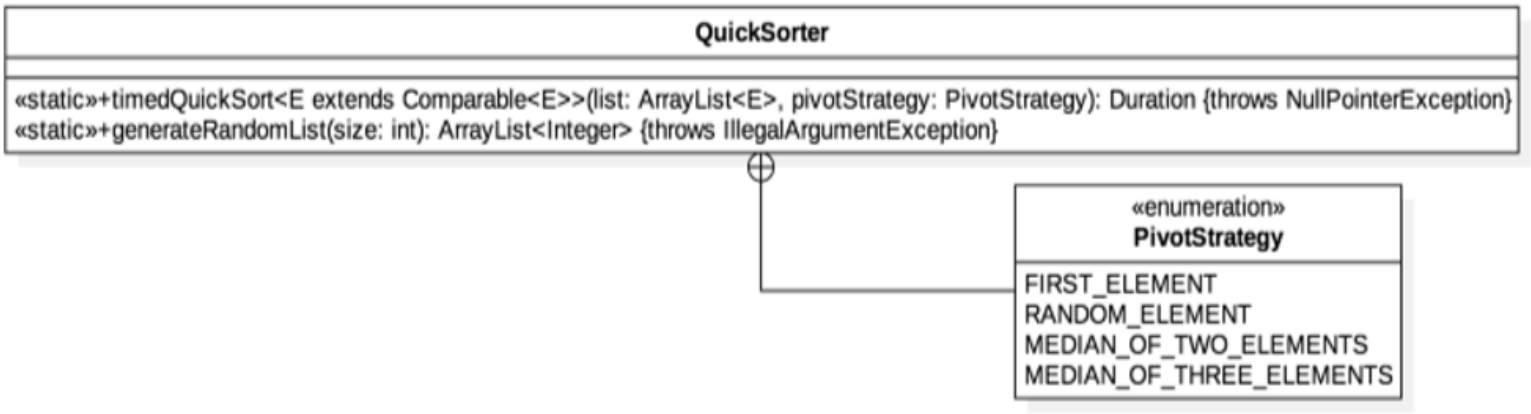
Project 5

QuickSort

The task of this project is to implement in Java several variations of the in-place QuickSort algorithm, each with a different choice of pivot. You should note the impact on execution time of different pivot selection strategies.

Specification

The java file name will be QuickSorter and it should have the following specifications.



`QuickSorter` is a static utility class – as such, it should contain only static members and should never be instantiated as an object. The `PivotStrategy` enumeration should be exactly bundled with (i.e. contained within) `QuickSorter`, but should be publicly accessible since access to it is required for clients to invoke the `timedQuickSort` method. The `timedQuickSort` method is a generic method that has a type parameter extending `java.lang.Comparable`, and that returns an instance of `java.time.Duration`.

In adddition to the `QuickSorter` java class, you will have another class that has the main function that takes four command line arguments. The first argument is the array size the second one is filename for the reports file, the third one is the filename to store the unsorted array and the fourth is the filename to store sorted array. The command line execution will look like the following:

./Main_Class 100000 report.txt unsorted.txt sorted.txt

Behavior

The program should sort any array using in-place quicksort and the following four different pivot selection strategies:

- FIRST_ELEMENT :- First element as pivot
- RANDOM_ELEMENT :- Randomly choosing the pivot element
- MEDIAN_OF_THREE_RANDOM_ELEMENTS :- Choosing the median of 3 randomly chosen elements as the pivot
- MEDIAN_OF_THREE_ELEMENTS :- Median of first, center and last element (book method).

`timedQuickSort` should accept an array-based list, sort it in-place using the QuickSort algorithm with the different pivot selection strategy , and return the time in nanoseconds that it took to sort the list. This method should immediately throw a `NullPointerException` with an appropriate message if either argument is null. This method should not throw any other exceptions given the above specification. You may assume that the list argument is in a valid state and that every element therein is non-null; however, you should not assume that the list argument is non-empty.

The `QuickSort` should be "in-place," meaning that you should not make a copy of the list, but rather sort the original list itself. It is recommended that you modularize the `QuickSort` algorithm and create separate private helper methods corresponding to each of the pivot selection strategies. You should time the actual sort itself using `java.lang.System.nanoTime()`. Remember the quicksort will use the recursive strategy and for small array (<20 elements) use insertion sort methodology.

`generateRandomList` generates and returns a new integer array-based list with the given size (i.e. length) that consists of random and unsorted values. The given size should be nonnegative, and this method should throw an `IllegalArgumentException` otherwise. This method should not throw any other exceptions. The returned list should be non-null, as should every element therein. You should use `java.util.Random` to generate integral values uniformly across the entire range of the `int` data type.

You will have a main function that reads the four command line arguments. First, it generates a integer list of specified size using the `QuickSorter.generateRandomlist` function. Then calls the several `QuickSorter.timedQuickSort` function with the same list and different pivot strategies. Before any sorting begins, record the unsorted array into a new file created with the name in the third place in the command line. After all sorting is over record the final sorted array into a new file created with the name in the fourth place in command line.

The `timedQuickSort` function will sort the list using given pivot selection strategies and then return the time in nano seconds to the main functions. Then the main function will write the result into a new file with the filename equal to second command line argument

(corresponding to “report.txt” in the example command line). The report file will have exactly five lines as shown in the example.

The report file format is as follows:

```
Array Size = 500000
FIRST_ELEMENT : PT0.224627921S
RANDOM_ELEMENT : PT0.231300908S
MEDIAN_OF_THREE_RANDOM _ELEMENTS : PT0.302695249S
MEDIAN_OF_THREE_ELEMENTS : PT0.317683418S
```

Generate reports for random arrays of different sizes. For certain instances generate the random array and sort the array using `java.util.Arrays.sort(int [])` and then call the `timedQuickSort` to observe its preformation for different pivot strategies for already sorted arrays. In addition, you should try for arrays that are almost sorted, to generate almost sorted arrays, first generate the array then sort it and then randomly swap 10 % of array elements. These results should be recorded and summarized in your readme file.

Submission

Submit the following items on eLearning :

1. README.txt

   This should identify who you are (name, NetID, etc.), which project you are submitting, what files comprise your project, how you developed and compiled your project (e.g. what IDE or text editor, which version of Java, what compiler options, etc.), and any other information you believe the grader should know or you want the grader to know. Along with the information include copies of several report files that were generated by your program for different array sizes. Also summarize in your words which pivot strategy was the fastest and why in your opinion. Also summarize the results for sorted and almost sorted arrays and which pivot strategy was faster for these scenarios.

2. QuickSorter.java

   This should be the only source code file that you submit. It should not include a main method. It should have the above stated public methods (it can have any number of private helper methods). Have the file inside default package.

3. Main class

   Give an appropriate name and indicate it in the readme file. Should take four command line arguments and handle all file IO operations accordingly. Have the file inside default package.

Both items should be submitted as a single zipped file named with your lowercase NetID. The file structure should resemble the following example:

*-- abc123789.zip
|-- README.txt
|-- QuickSorter.java
|-- Main.java

Evaluation

This project will be evaluated primarily according to the correctness of your QuickSort implementation for each of the four pivot selection strategies. Incorrect implementations may still be awarded partial credit based on how "sorted" your output list is according to some standard metric, e.g. inversion number. However, incorrect implementations that result in crashes, that do not attempt to implement the specified pivot selection strategy, etc. may not be awarded any credit. Regarding timing the sort duration, this aspect will be evaluated only on correct implementation and general "reasonableness" of your times – you are not required to achieve any speed benchmarks. Note that for very small list sizes, the empirically measured duration of your sort may appear identical for different pivot selection strategies – this may be because your physical machine may not support individual nanosecond resolution. Note as well that inferior pivot selection strategies may occasionally perform better than superior pivot selection strategies – this is due to the nature of randomness. The rubric is as follows:

Category	Weight
FIRST_ELEMENT QuickSort strategy	20%
RANDOM_ELEMENT QuickSort strategy	20%
MEDIAN_OF_TWO_ELEMENTS QuickSort strategy	20%
MEDIAN_OF_THREE_ELEMENTS QuickSort strategy	20%
QuickSort time measurement	10%
generateRandomList(int)	5%
Report	5%