# Project 5

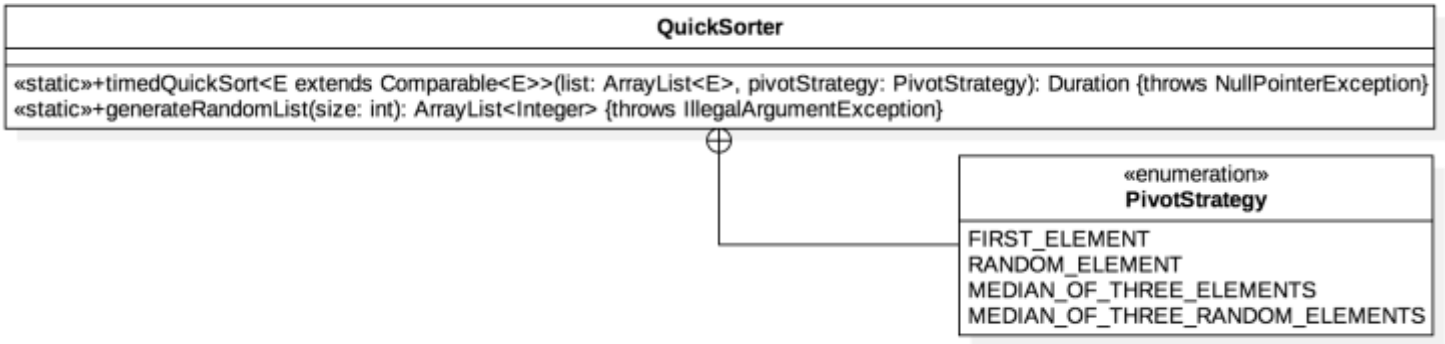
QuickSort

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 project must implement the following specification exactly, including all identifier names, method signatures, the presence or absence of exceptional behavior, etc. That being said, anything not clearly indicated by the UML diagram(s) or explicitly stated in the description is left up to your discretion. You may also add private helper methods or additional fields as necessary.

### Structure




`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.

For this project, your implementation should **not** rely on the default package – rather you should locate your project in a package exactly named "edu.utdallas.cs3345.project5". This name emulates the official Java package naming conventions, if you're curious.

### Behavior

`timedQuickSort` should accept an array-based list, sort it *in-place* using the QuickSort algorithm with the specified 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.[1] 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 return value of this method should be non-null.

The QuickSort should be "in-place," meaning that you should *not* make a copy of the list, but rather sort the original list itself.[2] 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().

`generateRandomList` generates and returns a new array-based list with the given size (i.e. length) that consists of random and unsorted integral values. The given size should be non-negative, 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.

`PivotStrategy` describes four pivot selection strategies for the QuickSort algorithm. The `FIRST_ELEMENT` strategy always selects the first element of the (sub-)array to use as the pivot on each iteration. The `RANDOM_ELEMENT` strategy selects a random element of the (sub-)array to use as the pivot on each iteration. The `MEDIAN_OF_THREE_ELEMENTS` strategy fetches the first, middle, and last elements of the (sub-)array on each iteration, and uses the median of those three values as the pivot. The `MEDIAN_OF_THREE_RANDOM_ELEMENTS` strategy fetches three random elements of the (sub-)array on each iteration, and uses the median of those three values as the pivot. The `PivotStrategy` enumeration may be a simple enum type – it need not contain any members beyond the enumeration of values.

## Submission

Submit the following items on eLearning before **11:59PM on Sunday, November 12th, 2017**:

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.
2. **QuickSorter.java**
   This should be the only source code file that you submit. It should not include a `main` method, since it is intended to be used like library code. (that is also how it will be tested) The source code should include an appropriate Javadoc for every public method and class. The `PivotStrategy` enumeration should also be included in this file.

---

[1] You may optionally choose to throw an `IllegalArgumentException` if the `pivotStrategy` argument value is not one of the four specified – this may be done "in the real world" to defend against future updates to the codebase that may add new `PivotStrategy` values, for instance. However, you will not be graded on this.

[2] Recall that object variables in Java are actually "references" (C++ calls them "pointers").

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
```

## 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 (i.e. Kendall Tau distance) or deletion 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. These factors will be taken into account for grading.

The project will be tested with multiple randomly generated lists for each pivot selection strategy. Bonus points may be awarded for exceptional code quality, which comprises engineering and design (e.g. modularity), documentation and comments (e.g. Javadoc), and style and layout (e.g. visually grouping lines into logical "paragraphs"). Penalties may be applied for not adhering to the specification, but will not be applied for otherwise poor code quality.

The rubric is as follows:

| Category | Weight |
|---|---|
| `FIRST_ELEMENT` QuickSort strategy | 20% |
| `RANDOM_ELEMENT` QuickSort strategy | 20% |
| `MEDIAN_OF_THREE_ELEMENTS` QuickSort strategy | 20% |
| `MEDIAN_OF_THREE_RANDOM_ELEMENTS` QuickSort strategy | 20% |
| QuickSort time measurement | 10% |
| `generateRandomList(int)` | 5% |
| Input Validation | 5% |