Alea Software Assignment

In this application, I supposed to get a generated random number that should be between 0 and 10 (1-digit number) for simplicity.

The algorithm is like the following:

1) Generate 1 million random numbers.
2) Write them to file
3) Sort the numbers
4) Write them to file

In java 8 we can use **Random**, **SplittableRandom**, **SecureRandom** and **ThreadLocalRandom** classes to generate streams of random numbers.

```
IntStream numberStream = new Random().ints(streamSize, 0, 10);

IntStream numberStream = new SplittableRandom().ints(streamSize, 0, 10);
```

According to following links SplittableRandom is faster than Random class
https://www.logicbig.com/how-to/java-random/different-random-classes.html

https://stackoverflow.com/questions/29193371/fast-real-valued-random-generator-in-java

for generating one million numbers I got the following results:
Random: 4ms
SplittableRandom: 3ms

Using stream of random numbers for our application, we have a situation, Java streams were designed to be operated only once, if we save it to a file we need to find a solution to keep data in memory and sort it and then save it in another file.

1) The first approach to solve this problem is to write the data to file and then read it from file and sort it and then write it to another file. This approach has I/O performance issue because we have to read the data from file for second step.

2) The second approach is convert the stream numbers into an array and then get a new Stream from that array whenever we want to do an operation on it.
   We can use a new feature in java8 to sort arrays, parallelSort() .

   The parallelSort() method uses the concept of multithreading which makes it much faster compared to the normal sort when there are lot of elements.

   The weakness of this approach is, we have memory issue for large amount of data, but comparing with first approach, this is better.

```
int[] intArray = numberStream.toArray();
```

```
IntStream numberStream = Arrays.stream(intArray);


Arrays.parallelSort(intArray); // sort array

IntStream numberStreamSorted = Arrays.stream(intArray);

writeToFile(filePath, numberStream);

writeToFile(randNumPath, numberStreamSorted);
```

time of generating and converting to array,
Random: 130ms
SplittableRandom: 110ms.

So we choose SplittableRandom for further tests

```
Whole operation :401ms to complete.
```

3) What if we generate numbers and keep them in array from beginning, instead of converting the stream to array? I tested this approach the result was much better:

   Time is: 20ms, comparing with previous approach is much better.

   The following link also mention that stream for-each is slower than for-loop:
   https://blog.jooq.org/2015/12/08/3-reasons-why-you-shouldnt-replace-your-for-loops-by-stream-foreach/

```
Whole operation :141ms to complete.
```

Since approach 1 is worse than 2 others, I just compared approach 2 and approach 3.