Task

- ✓ Create a class `ProbabilityBucketSort` under the `sort` package.
  - ✓ Inherit the `AbstractBucketSort<Double>` class.
- ✓ Define a constructor.
  - ✓ Call `super` with an appropriate bucket size given the input range of [0, 1), and `sort = true`.
- ✓ Override the `getBucketIndex` method.
  - ✓ `@return` the index of the bucket that `key`∈[0, 1) should be inserted.
- ✓ Create a class `ProbabilityBucketSortTest` under the `sort` package, and define a method `test` to evaluate the accuracy of your algorithm using 10 different lists of 100 random double values between 0 and 1.
  - ✓ See `SortTest#testAccuracy` and `Random#nextDouble`.

Solution

See java files "ProbabilityBucketSort.java" and "ProbabilityBucketSortTest.java" for code.

The purpose of the test file is to create a random Double array, copy it and then sort one of the arrays with my ProbabilityBucketSort (passed through and labeled engine) and the other with Array.sort().

The test file asks for how large of an array you want randomized, and how many times you'd like to run the test. In this case we wanted 10 iterations on arrays the size of 100.

Using Junit, the test compares both arrays and returns whether they are equal or not. They would be equal if ProbabilityBucketSort indeed worked and sorted the array labeled "`original`".

My sorting method ProbabilityBucketSort, creates 10 buckets.

Index	0	1	2	3	4	5	6	7	8	9
Values	[0-0.1)	[0.1-0.2)	[0.2-0.3)	[0.3-0.4)	[0.4-0.5)	[0.5-0.6)	[0.6-0.7)	[0.7-0.8)	[0.8-0.9)	[0.9-1)

I believe this is reasonable as the array we will be sorting was populated with random Doubles, meaning it is fair to assume an even distribution of keys from 0 to 1.

Then calls on super.sort to assign/sort elements from the 100 size array into buckets using the method getBucketIndex() to determine the index of the bucket that the key belongs to.

super.sort continues using AbstractBucketSort.sort to sort the buckets and spill them out.