Implementing List Collections with Performance Constraints

Problem Description

This assignment focuses on applying the two collection implementation patterns that we've discussed: use data structures based on static memory (arrays) or dynamic memory (linked nodes). The collections we're dealing with are two very different List collections that are specified in a small interface hierarchy. In addition to implementing the functional specification (i.e., the method behavior), you must also meet non-functional performance constraints.

As always, you must closely adhere to the API as specified in the provided interfaces. Deviation from any aspect of the API specified in these interfaces, or the items listed below, will result in a significant deduction of points.

Performance Constraints

The interface methods that you implement must meet the following performance constraints expressed in terms of big-oh. Remember that big-oh is an *upper bound*; therefore, your actual performance can perhaps be faster.

- **RandomizedList** — All interface methods must have $O(1)$ worst-case time complexity, except for `iterator`, which can be $O(N)$. The iterator methods themselves (`hasNext()`, `next()`) must be $O(1)$. Amortization is allowable in $O(1)$ times. The data structure that you implement must use memory proportional to the current number of elements in the list. Each iterator created by this class can also use a linear amount of memory with respect the number of elements in the list.
- **DoubleEndedList** — All interface methods, with no exceptions, must have $O(1)$ worst-case time complexity, but amortization is allowable. The data structure that you implement must use memory proportional to the current number of elements in the list. Each iterator created by this class can only use a constant amount of memory, not a linear amount.

API and Other Constraints

You have been provided the code for the interfaces and a factory class[1]. While you have a measure of freedom in exactly how you design and build the implementing classes, you are subject to the following constraints.

- You must not change (add, delete, modify) any provided interface in any way.
- You can only change the factory class by filling in a correct return value in the factory methods. You may not make any other changes to that class.
- The implementing classes that you create must be based on either an array data structure or a linked chain of nodes.

[1] `https://en.wikipedia.org/wiki/Factory_method_pattern`

- You may not use any pre-built collection (e.g., a JCF class or a collection implemented in a textbook or online). The use of any pre-built collection will result in a grade of zero points. This assignment is all about you learning to build collections from first principles (using arrays and chains of linked nodes).
- The only imports you can use in any code that you write are for java.util.Iterator, java.util.Random, and java.util.NoSuchElementException. No other imports are allowed, and you may not circumvent this constraint by using fully qualified names.
- You will need to create additional helper classes (e.g., for iteration, nodes, etc.), but they must be implemented as nested classes and not top-level classes. You can write only two top-level classes: one that implements the RandomizedList interface and one that implements the DoubleEndedList interface. No additional top-level classes can be used.
- The remove method of an iterator must not remove any element of the underlying list but should throw an UnsupportedOperationException.
- The next method of an iterator must throw a NoSuchElementException if there are no more elements in the iteration.
- The only warning that your code is allowed to generate is the warning that results from casting a newly allocated array to a generic type. You may use the @SuppressWarnings annotation to keep this from being reported, if you wish. No other warnings are allowed.

Notes and other requirements

- Read this handout carefully. Read the provided source code carefully. Ask questions on Piazza. Start early and be proactive.
- Your submission must be solely your own work. Discussing ideas and general approaches to a problem is fine, but sharing source code, even small sections of it, is considered a violation of the academic honesty policy.