Project 3: An Adaptive List Implementation

1. Description

This assignment gives you an opportunity to work with a list data structure by implementing Java API's interfaces `public inteface List<E>` and `public inteface ListIterator<E>` both under `java.util` package using an adaptive list. This adaptive list is a double linked list complemented by an array for indexed read operations (like `get(int pos)`) and indexed write operations (like `set(int pos, E obj)`). This adaptive list is much more efficient than the doubly linked list for supporting a long sequence of indexed read/write operations flanked by `add(...)` and `remove(...)` operations.

2. Requirements

Write a generic linked list class named `AdaptiveList`. Your class must implement the `java.util.List` interface. You may find it helpful to read the Java API 8's Javadoc for the interface. All the methods except `subList` method in the interface methods must be implemented without throwing an `UnsupportedOperationException`, i.e., you may throw an `UnsupportedOperationException` for the `subList` method. Note that for some of the methods, we provide their implementations as examples for you to study or for showing the list/array created by your code; you just need to implement the other methods with the comment line `//TODO` in the body. You are not allowed to use any `Collection` class in your implementation. Usage of import statements, as well as fully qualified class name, is not allowed. Creating and using your own custom classes is not allowed.

The `AdaptiveList` has a non-static inner class named `ListNode` whose instances serve as nodes in the doubly linked list. The inner class has three data fields: `data` of generic type `E`, `next` of type `ListNode`, and `prev` of type `ListNode`. Introduction of additional data fields in the `ListNode` class is not allowed, as well as in `AdaptiveList` and `AdaptiveListIterator` besides the ones that were already provided. The `next` data field of a node refers to its successor node in the list if the successor node exists and is `null` otherwise. The `prev` data field of a node refers to its predecessor node in the list if the predecessor node exists and is `null` otherwise. Every `AdaptiveList` list must have two dummy nodes named `head` and `tail` along with data nodes in the chain. The list is empty if it has no data nodes. If the list is empty, then `head` is the predecessor of `tail` and `tail` is the successor of `head`. Otherwise, `head` is the predecessor of the first data node and the first data node is the successor node of `head`; `tail` is the successor of the last data node and the last data node is the predecessor of `tail`. The `prev` data field of `head` and the `next` data field of `tail` are always `null`, as well as `data` data field.

Write a private inner class named `AdaptiveListIterator` to implement the `ListIterator` interface. You should implement all methods in the `ListIterator` interface without throwing any `UnsupportedOperationException`. There is no need to keep a coherent list if there is concurrent modifiction to the list. In other words, the iterator does not have to be a fail-fast iterator.

In addition to the doubly linked list, the `AdaptiveList` class keeps an array of type `E` elements for implementing the `get(int pos)`, `set(int pos, E obj)`, `reorderEvenOdd()`, and `reverse()` methods efficiently. Note that `reverse()` method swaps the elements at indices 0 and $n-1$, at indices 1 and $n-2$, and so on, so that the order of the elements in the array of length n is reversed. The method returns `false` if $n <= 1$ and `true` oth-

erwise. The method `reorderEvenOdd()` swaps elements at even positioned indices with the ones in subsequent odd positioned indices, i.e., 0 with 1, 2 with 3, and so on. If the length is odd then last element stays at its position. This method return `false` if $n <= 1$ and `true` otherwise. Both `reverse()` and `reorderEvenOdd()` methods need to be implemented without using any additional arrays.

The doubly linked list and the array are alternately used as follows. The class keeps two boolean data fields named `linkedUTD` and `arrayUTD`, where `UTD` stands for **Up To Date**: `linkedUTD` is `true` if the doubly linked list is used to represent the current sequence of data items and `false` otherwise; `arrayUTD` is `true` if the array is used to represent the current sequence of data items and `false` otherwise. At any time, the current sequence of data items is represented either by doubly linked list or by the array; so at least one of `linkedUTD` or `arrayUTD` is `true`. The doubly linked list is used to implement all methods except for the `get(int pos)`, `set(int pos, E obj)`, `reverse()`, and `reorderEvenOdd()` methods, which are implemented by using the array. These implementations are facilitated by using two helper methods: The `updateLinked()` method creates a new doubly linked list with `numItems` data nodes by copying the current sequence of data items from the array to the doubly linked list and setting `linkedUTD` to `true`, whereas the `updateArray()` method creates a new array of length `numItems` by copying the current sequence of data items from the doubly linked list to the array and setting `arrayUTD` to `true`. If a method is to be implemented by using the doubly linked list but `linkedUTD` is `false`, then `updateLinked()` needs to be called before the implementation and at the end `arrayUTD` needs to be set to `false` if the doubly linked list is modified by the method so that the array is no longer up to date. Similarly, if a method is to be implemented by using the array but `arrayUTD` is `false`, then `updateArray()` needs to be called before the implementation and at the end `linkedUTD` needs to be set to `false` if the array is to be modified by the `set(int pos, E obj)`, `reverse()`, or `reorderOddEven()` methods (see the examples below).

```
AdaptiveList<String> seq = new AdaptiveList<String>();
seq.add("B");
seq.add("A");
seq.add("C");
System.out.println("After the three seq.add()
  operations:");
System.out.println("linkedUTD: " + seq.getlinkedUTD());
System.out.println("arrayUTD: " + seq.getarrayUTD());
System.out.println(seq.toString());
System.out.println( seq.get(1) );
System.out.println("After the seq.get(1) operation:");
System.out.println("linkedUTD: " + seq.getlinkedUTD());
System.out.println("arrayUTD: " + seq.getarrayUTD());
System.out.println(seq.toString());
System.out.println( seq.set(1, "D") );
System.out.println("After the seq.set(1, 'D')
  operation:");
System.out.println("linkedUTD: " + seq.getlinkedUTD());
System.out.println("arrayUTD: " + seq.getarrayUTD());
System.out.println(seq.toString());
seq.add("E");
System.out.println("After the seq.add('E') operation:");
System.out.println("linkedUTD: " + seq.getlinkedUTD());
System.out.println("arrayUTD: " + seq.getarrayUTD());
System.out.println(seq.toString());
```

Output would be as follows:

```
After the three seq.add() operations:
linkedUTD: true
arrayUTD: false
A sequence of items from the most recent array:
[]
A sequence of items from the most recent ↩
  linked list:
(B, A, C)
```

```
A
After the seq.get(1) operation:
linkedUTD: true
arrayUTD: true
A sequence of items from the most recent array:
[B, A, C]
A sequence of items from the most recent ↩
  linked list:
(B, A, C)
A
After the seq.set(1, 'D') operation:
linkedUTD: false
arrayUTD: true
A sequence of items from the most recent array:
[B, D, C]
A sequence of items from the most recent ↩
  linked list:
(B, A, C)
After the seq.add('E') operation:
linkedUTD: true
arrayUTD: false
A sequence of items from the most recent array:
[B, D, C]
A sequence of items from the most recent ↩
  linked list:
(B, D, C, E)
```

3. Submission

You are required to include, in your submission, the source code for each of the classes in the code template, as well as any additional methods you may have written to complete the project. In each of the class files you need to put your Firstname and Lastname after the Javadoc *@author* tag. **In case you introduce new (helper) methods then for each of these you need to write a proper Javadoc style comments.** Write your project classes in the `edu.iastate.cs228.proj3` package. **TEST SCRIPTS ARE REQUIRED for this project, and it needs to be named** `TestAdaptiveList`. Your **.zip** file should be named `Firstname_Lastname_PROJ3.zip`, where `Firstname` needs to be replaced with your first name, and `Lastname` needs to be replaced with your last name, and in both cases only first letter needs to be capital and all other subsequent letters need to be in lower case. If your first and/or last names include hyphens, then for each part where there is a hyphen do as follows: e.g., if your last name is Abc-Def, then for Lastname it needs to be Abc_Def.

Note 0: Make sure to download your submission after you upload your solutions, and unzip it and make sure that you have in there the structured directories and inside you have the actual source codes, i.e., `.java` files. **If we do not find actual source files then you would be getting** 0 **for this project.** We strongly suggest you to upload your partially ready solutions as soon as they are ready. Since you are allowed to have unlimited submissions until the due date (and we are grading only your last submitted set of solutions). This way you

would still have partial credit in case you experience any technical issues towards the last minutes of due time. If you are not able to access Canvas, then feel free to send your solution files by email to the professor only, however, note that emails must be received by the due date and time.

Note 1: Since for projects you are allowed to submit any of your two projects 24 hours late without any penalties, there will be difference of a day on due and availability dates of the project. If you submit any solutions files after due date but by availability date, then we will automatically count it towards any of your unused 24 hour late submissions without any penalties. This also means that you cannot ask teaching staff to consider any of your previous submissions so that your no penalty attempts could still be used for future projects. If you are out of no penalty attempts but you submit your solutions by the availability date (i.e., after the due date) then it is not accepted, i.e., you will get 0, of course, if you have not submitted any files before the due date. In case of latter, we will count your submission that is submitted by the due date. Note that it's perfectly fine to combine your two no penalty late submissions and reuse it to submit your project solutions 48 hours late without any penalties, however, note that in this case you will need to send the files by email to the professor.