Lab: `MyLinkedList<E>` Implementation

AP Computer Science with Data Structures

Background: A `toString` Method

A `toString()` method returns a `String` representation of the object that contains the `toString()` method. It is defined in class `Object` and its behavior is to return a `String` containing the memory address where the `Object` is stored. Typically, the author of a class overrides the `toString()` method. The `toString()` method for an object is called automatically whenever a `String` representation of the object is needed.

You may wish to override `toString()` in your class. As an example, here is the code for creating a `String` representation of an array named `values` of type `Object`. Notice that it relies on the polymorphic behavior of Java objects.

```
public String toString()
{
     if (size == 0)
          return "[]";

     String s = "[";
     for (int i = 0; i < size - 1; i++)
          s += values[i] + ",";
     return s + values[size - 1] + "]";
}
```

Exercise: Burning the `MyLinkedList<E>` at Both Ends

Create a Java Project to hold your `MyLinkedList` code. Within that project, create a package called `myLinkedList` (notice the capitalization, it is significant.) Your implementation of `MyLinkedList` and `DoubleNode` goes inside the `myLinkedList` package. Testers and other code **does not** go inside the `MyLinkedList` package.

Create a class called `DoubleNode` inside the `myLinkedList` package. Create a class called `MyLinkedList` inside the `myLinkedList` package. Implement your design for the `DoubleNode` object.

Implement the following methods using your design and the supplied `MyList,` and `MyListIterator` interface code. The `iterator` and `listIterator` methods should return null. Also implement any other methods you included in your design.

`Interface MyList<E>`

- `int size()` //returns the current `size` of the list
- `boolean add(E obj)` //appends `obj` to the end of the list;
  //returns true
- `void add(int index, E obj)` //inserts `obj` at position
  //`index`(0≤`index`≤`size`), moving elements
  //at position `index` and higher to the
  //right (adds one to their index) and
  //adjusts `size`
- `E get(int index)` // returns the element at position `index`
- `E set(int index, E obj)` //replaces the element at position `index` with `obj;`
  //returns the element formerly at the specified position
- `E remove(int index)` //removes element from position `index`, moving
  //elements at position `index+1` and higher to the left
  //(subtracts one from their index) and adjusts the size;
  //returns the element formerly at the specified position.
- `Iterator<E> iterator()`
- `MyListIterator<E> listIterator()`

When you have finished, test your design by writing a test program that implements your test plan. Note that you will need to import your `MyLinkedList` code by using the import statement: `import myLinkedList.*;`. You may optionally run the supplied tester (which your teacher will surely run to test your code).

Exercise: Putting it in a Jar

Once you have finished testing your `MyLinkedList`, construct a Java Jar library as follows:

1. In the directory that holds your jar library, which currently contains your `MyArrayList` implementation, create a sub-directory called 'myLinkedList'.
2. Copy the source code and the .class files from your project into the myLinkedList sub-directory. Do not include any testing code or anything that has a `main` method.
3. Launch the command (terminal) window and navigate to the directory that will hold your jar file (**not** the sub-directory).
4. Run the java jar utility using the following command:
   `jar –cvf MyLib.jar myArrayList/*, myLinkedList/*`
5. Verify that the jar file was created.

To test your library, create a new Java project, and import only the tester. Make sure that your jar file is included as an external jar file or library. Make sure the tester still runs.

What to Turn In

Turn in your jar file to Athena2. If you wrote your own tester, submit your tester as well.