Assignment 6

Reminder: Your code is to be designed and written by only you and not to be shared with anyone else. See the Course Outline for details explaining the policies on Academic Integrity. Submissions that violate the Academic Integrity policy will be forwarded directly to the Computer Science Academic Integrity Committee.
All materials provided to you for this work are copyrighted, these and all solutions you create for this work cannot be shared in any form (digital, printed or otherwise). Any violations of this will be investigated and reported to Academic Integrity.

Objectives

- More practice implementing linked list data structure
- Designing recursive methods

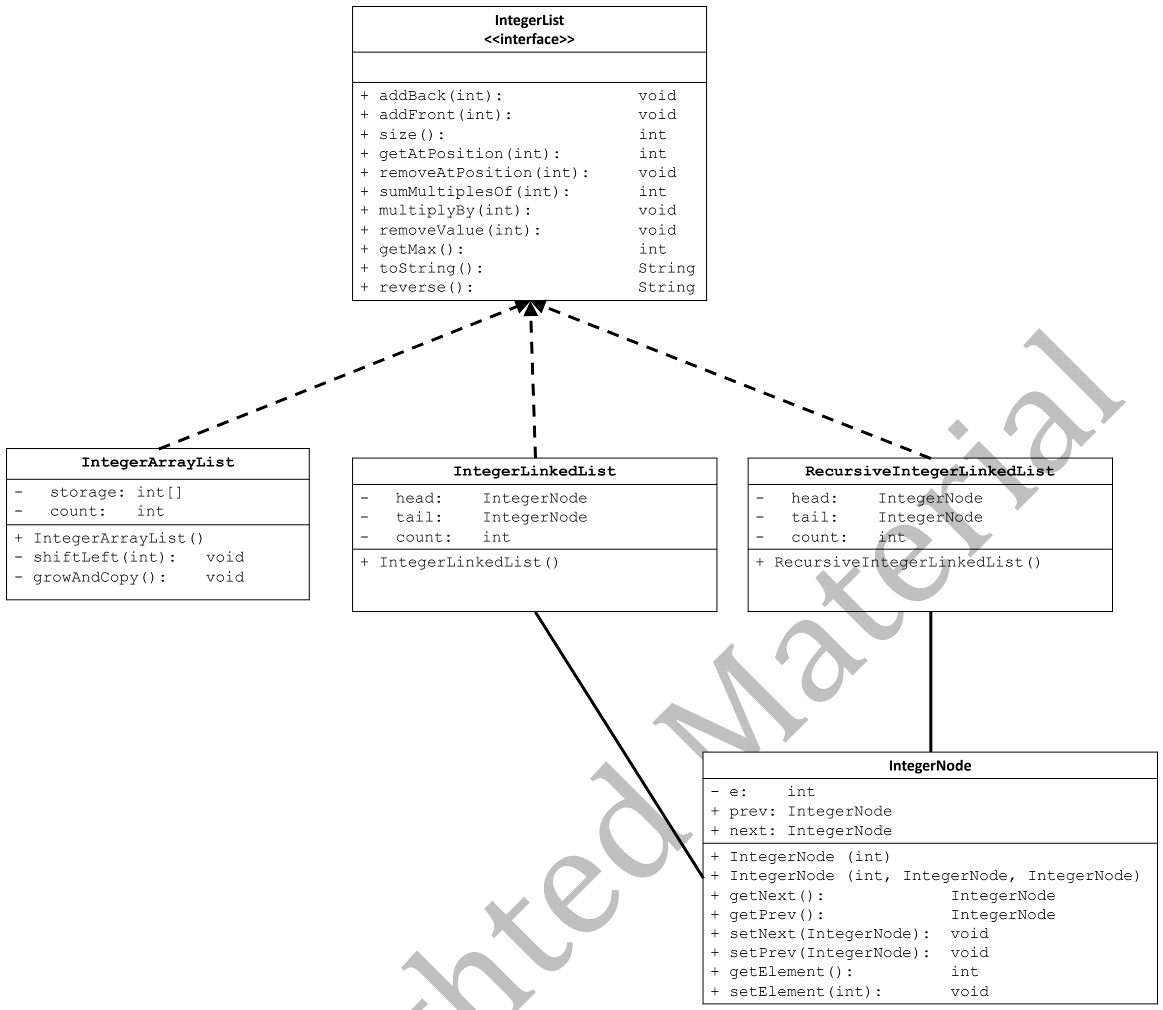
Introduction

This assignment will build on your knowledge of the list interface. Your assignment is to implement the `IntegerList` interface defined in `IntegerList.java` as a doubly-linked list in `RecursiveIntegerLinkedList.java` A UML overview is on the following page which also illustrates how we can have alternative array-based implementation of the `IntegerList` interface.

Wherever possible you **MUST** implement your methods using recursion. That is, any method that requires you to traverse the list must be implemented recursively, not iteratively. In Assignment 5 you implemented these methods iteratively in `IntegerLinkedList.java`. If you submit a file with ANY traversal method implemented iteratively, you will receive a **ZERO grade** for the whole assignment.

Some methods do not require traversal of the list and therefore the implementation of those methods will be the same in `RecursiveIntegerLinkedList.java` as they were in your `IntegerLinkedList.java`. We will not consider it plagiarism if you copy your solution to these methods from `IntegerLinkedList.java` to `RecursiveIntegerLinkedList.java`.

For example, the `size` method does not traverse the list, it simply returns the value of the `count` field/attribute therefore this method will be the same in `RecursiveIntegerLinkedList.java` and `IntegerLinkedList.java`



Submission and Grading

Submit your `RecursiveIntegerLinkedList.java` with the completed methods through the assignment link in BrightSpace.

- **CRITICAL:** If you submit a file with **one or more** traversal methods implemented iteratively, you will receive a **ZERO grade** for the **whole assignment**.
- You **must** name the methods in `RecursiveIntegerLinkedList.java` as specified in the given interface and as used in `A6Tester.java` or you will receive a **zero grade** as the tester will not compile.
- If you chose not to complete some of the methods required, you **must at least provide a stub for the incomplete method** in order for our tester to compile.
- If you submit files that do not compile with our tester (ie. an incorrect filename, missing method, etc) you will receive a **zero grade** for the assignment.
- Your code must **not** be written to specifically pass the test cases in the testers, instead, it must work on other inputs. We may change the input values when we run the tests and we will inspect your code for hard-coded solutions.
- **ALL late** and **incorrect** submissions will be given a **ZERO** grade.

Getting Started

1) Download all java files provided in the Assignment folder on BrightSpace.

2) Try to compile `A6Tester.java` You will see it does not compile because your `RecursiveIntegerLinkedList` class is missing the required methods. NOTE: we have provided you with the `toString` and `reverse` method implementations – DO NOT change these.

3) Introduce stubs for your constructor and for each of the methods `RecursiveIntegerLinkedList` must implement. **DO NOT** move on until you have the tester compiling with no errors!

4) Implement each method in `RecursiveIntegerLinkedList.java` by repeating the following until all of the test methods in `main` of `A6Tester.java` are uncommented and all tests pass.

   a) Uncomment one of the test methods in the `main` of `A6Tester.java`

   b) Implement ***one*** of the methods being tested in `RecursiveIntegerLinkedList.java`

   c) Compile and run the test program `A6Tester.java`

Experiment with the interface:

Interested in experimenting with changing the list type?

Update the `createList` method in A6Tester.java, adding the bolded code below:

```
public static IntegerList createList() {
    if (listType.equals("linked")){
        return new IntegerLinkedList();
    } else if (listType.equals("recursive")){
        return new RecursiveIntegerLinkedList();
    } else {
        System.out.println("list type specified not supported");
        return null;
    }
}
```

Notice in the `main` method, the following conditional code:

```
if (args.length == 1)
    listType = args[0];
```

This allows us to compile our tester once with: `javac A6Tester.java`

And then we can run the program giving it either `linked` or `recursive` as a command line argument, determining at runtime what kind of list we want to use as shown below. Notice: we don't have to recompile between these runs.

```
java A6Tester linked
java A6Tester recursive
```