Linked Lists

There are two kinds of linked lists: singly linked and doubly linked. You will start your algo data structures library by building these two types of linked lists.

# Requirements

Start by building a new library to contain all of your future data structures. To start this library off, your goal is to build two linked list data structures that meet the following requirements:

1. You may build a DLL in C# or a JAR in Java (your choice).
2. Your namespace/package must be named EXACTLY as follows:  
   C# namespace: AlgoDataStructures  
   Java package: algo.data.structures
3. Add two classes: SingleLinkedList and DoubleLinkedList
4. Each linked list class will be composed of nodes. You have autonomy in implementing this part of the architecture
5. Both list classes must be generic, capable of storing any kind of data within its nodes.
6. Each list class should have the following functions
   1. **Add**(T val) – puts a new value at the Tail end of the list
   2. **Insert**(T val, int index) – inserts a new value at a given index, pushing the existing value at that index to the next index spot, and so on. Insert may ONLY target indices that are currently in use. In other words, if you have 5 elements in your list, you may insert at any index between 0 and 4 inclusive. Index 5 would be considered out of bounds as it is not currently in use during the insertion process. Any index less than zero or equal to or greater than Count should throw an index out of bounds exception.
   3. **Count** – returns the number of values in the list. This should be a property in C#, but a method in Java. For efficiency, the count value should be a managed value (stored and updated as needed) and not simply derived each time Count is called.
   4. **Get**(int index) – returns the value at the given index. Any index less than zero or equal to or greater than Count should throw an index out of bounds exception.
   5. **Remove** – removes and returns the first value in the list
   6. **RemoveAt**(int index) –removes and returns the value at a given index. Any index less than zero or equal to or greater than Count should throw an index out of bounds exception.
   7. **RemoveLast** – removes and returns the last value in the list
   8. **ToString** – an override method that creates and returns a string representation of all the values in the list. The string must be in the format of “v0, v1, v2, .., vn-1” wheren-1 is the last index in the list. An empty list should return an empty string (but not null). While every value in the string is separated by a comma and space, the string must NOT have any unnecessary commas or spaces at the beginning or end.
   9. **Clear** – removes all values in the list.
   10. **Search**(T val) – searches for a value in the list and returns the first index of that value when found. If the key is not found in the list, the method returns -1.
7. Any function that adds or removes values from the list MUST impact Count accordingly.
8. Your DLL or JAR must follow a similar convention to LastF\_AlgoDataStructures (where Last is replaced with your last name and F is replaced with your first initial).

You are REQUIRED to follow the signatures as outlined above. This means that you MUST name the functions and order the parameter lists as I have. It should also be clear by the descriptions what return types (if any) are required. With that said, it is expected that you will follow the proper convention for your coding language (Pascal Casing for C#, Camel Casing for Java, etc). The functions are named as they would be in C#, but in Java, each name would begin with a lowercase letter. Do not deviate from these conventions.

# Rubric

**Automatic Zero:** Your data structures do not follow their respective proper architecture, you deviate from the prescribed function signatures, your final product is not a library (JAR or DLL), or you fail to name your library correctly.

(50 points) Your Linked List is implemented correctly and the functions work correctly (points for each passing test)