1. **Learning Resources:**
   * If you’re interested in diving deeper, there are plenty of tutorials and resources available:
     + [W3Schools Node.js Tutorial: Provides examples and explanations for beginners1](https://www.w3schools.com/nodejs/).
     + [Java Code Geeks Java Node Example: Demonstrates a simple Java Node class2](https://examples.javacodegeeks.com/java-node-example/).
     + [Stackify Node.js Tutorials: Covers installation, advanced concepts, and interview questions3](https://stackify.com/learn-nodejs-tutorials/).
     + [JavaTpoint Node.js Tutorial: Comprehensive tutorial covering various Node.js topics4](https://www.javatpoint.com/nodejs-tutorial).

Step One:

DSEList class:

1. **Constructor:**
   * Implement a blank constructor that takes no parameters.
   * This constructor initializes an empty DSEList.

* It have two constructors:
  + The blank constructor (public DSEList()) initializes an empty list.
  + The constructor with a Node parameter (public DSEList(Node head\_)) sets the head of the list to the given Node.
* For the blank constructor, you don’t need to do anything special since the default values (null for head and tail) are already set.
* For the constructor with a Node parameter, you can set the head to the provided Node.

1. **Constructor (Accepting a Node):**
   * Implement a constructor that accepts one Node containing a String object.
   * Set the head of the DSEList to the given Node.
2. **Copy Constructor:**
   * Implement a copy constructor that accepts a DSEList object.
   * Perform a deep copy of the DSEList passed to the constructor.
   * The new DSEList should not contain references to the Node objects in the second DSEList.
   * Ensure that changes to the contents of Node objects in one DSEList do not affect the other.

* The copy constructor (public DSEList(DSEList other)) creates a deep copy of the other list.
* You need to iterate through the other list and add each element to your new list.
* Make sure that the new list does not share references with the original list (i.e., modify the Node objects).

1. add(String obj)**Method:**
   * Append the specified String object to the end of the list.
   * Return true if the addition is successful.
2. isEmpty()**Method:**
   * Check whether the list is empty.
   * Return true if the list contains no elements.
3. size()**Method:**
   * Return the number of elements in the list.
4. toString()**Method:**
   * Create a String by concatenating each Node’s toString().
   * Insert a single space (’ ') between each Node’s toString().
   * No trailing space should be inserted.
   * Example: If the list contains 3 Node objects, an appropriate toString() return value could be '1 2 3'.
5. equals(Object other)**Method:**
   * Determine whether two DSEList objects are equal.
   * They are equal if they contain the same Strings in the same order.
6. remove(int index)**Method:**
   * Remove the String at the specified index.
   * Adjust the head and tail pointers if necessary.
7. indexOf(String obj)**Method:**
   * Search for the index of the given String in the list.
   * Return -1 if not found.
8. get(int index)**Method:**
   * Retrieve the String at the specified index.
   * Return null if the index is out of bounds.
9. add(int index, String obj)**Method:**
   * Add the given String at the specified index.
   * Adjust the pointers accordingly.
10. contains(String obj)**Method:**
    * Check if the list contains the given String.
    * Return true if found.
11. remove(String obj)**Method:**
    * Remove the specified String from the list.
    * Update the pointers.
12. hashCode()**Method:**
    * You can return a constant value (e.g., 0) for now.