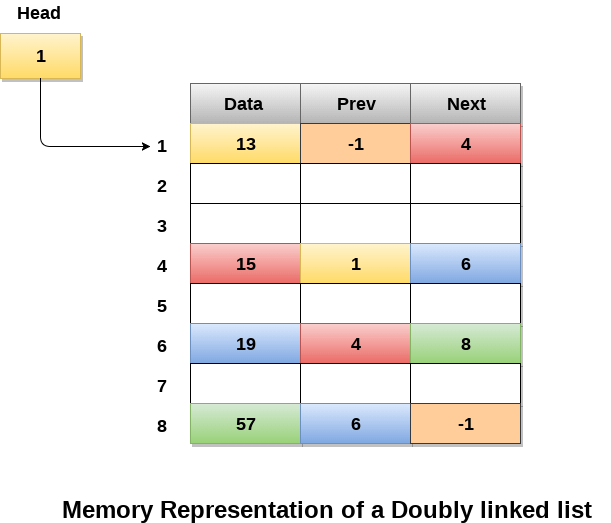
1. **Learning Resources:**
   * If you’re interested in diving deeper, there are plenty of tutorials and resources available:
     + [W3Schools Node.js Tutorial: Provides examples and explanations for beginners1](https://www.w3schools.com/nodejs/).
     + [Java Code Geeks Java Node Example: Demonstrates 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).

Doubly linked list is a complex type of linked list in which a node contains a pointer to the previous as well as the next node in the sequence. Therefore, in a doubly linked list, a node consists of three parts: node data, pointer to the next node in sequence (next pointer) , pointer to the previous node (previous pointer). A sample node in a doubly linked list is shown in the figure.



A doubly linked list containing three nodes having numbers from 1 to 3 in their data part, is shown in the following image.

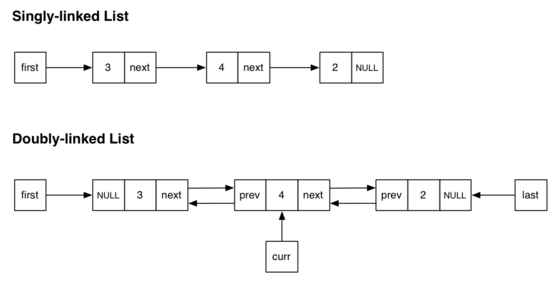




|  |  |  |
| --- | --- | --- |
| **SN** | **Operation** | **Description** |
| 1 | [Insertion at beginning](https://www.javatpoint.com/insertion-in-doubly-linked-list-at-beginning) | Adding the node into the linked list at beginning. |
| 2 | [Insertion at end](https://www.javatpoint.com/insertion-in-doubly-linked-list-at-the-end) | Adding the node into the linked list to the end. |
| 3 | [Insertion after specified node](https://www.javatpoint.com/insertion-in-doubly-linked-list-after-specified-node) | Adding the node into the linked list after the specified node. |
| 4 | [Deletion at beginning](https://www.javatpoint.com/deletion-in-doubly-linked-list-at-beginning) | Removing the node from beginning of the list |
| 5 | [Deletion at the end](https://www.javatpoint.com/deletion-in-doubly-linked-list-at-the-end) | Removing the node from end of the list. |
| 6 | [Deletion of the node having given data](https://www.javatpoint.com/deletion-in-doubly-linked-list-after-the-specified-node) | Removing the node which is present just after the node containing the given data. |
| 7 | [Searching](https://www.javatpoint.com/searching-in-doubly-linked-list) | Comparing each node data with the item to be searched and return the location of the item in the list if the item found else return null. |
| 8 | [Traversing](https://www.javatpoint.com/traversing-in-doubly-linked-list) | Visiting each node of the list at least once in order to perform some specific operation like searching, sorting, display, etc. |

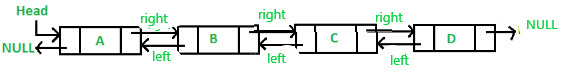
A diagram of a diagram

Description automatically generated



**Search an element in a Doubly Linked List**

Given a [Doubly linked list(DLL)](https://www.geeksforgeeks.org/doubly-linked-list/) containing **N** nodes and an integer **X**, the task is to find the position of the integer **X** in the doubly linked list. If no such position found then print **-1**.

[](https://media.geeksforgeeks.org/wp-content/uploads/20201125191410/rep.PNG)

**companies** is the name of a field in the **SecuritiesExchange** class, which is presumably a map (likely a **HashMap**) that stores **ListedCompany** objects. Let's break down what **companies.containsKey(company.getCode())** and **companies.put(company.getCode(), company)** are doing:

1. **companies.containsKey(company.getCode())**: This line checks if the **companies** map already contains a key corresponding to the code of the **ListedCompany** object passed as a parameter to the **addCompany** method. It uses **company.getCode()** to get the code of the provided **ListedCompany** object. If the map already contains the code as a key, it means that a company with the same code already exists in the map.
2. **companies.put(company.getCode(), company)**: This line adds the **ListedCompany** object passed as a parameter to the **addCompany** method to the **companies** map. It uses **company.getCode()** to get the code of the provided **ListedCompany** object, which serves as the key in the map. The entire **ListedCompany** object is then stored in the map with the code as its key.