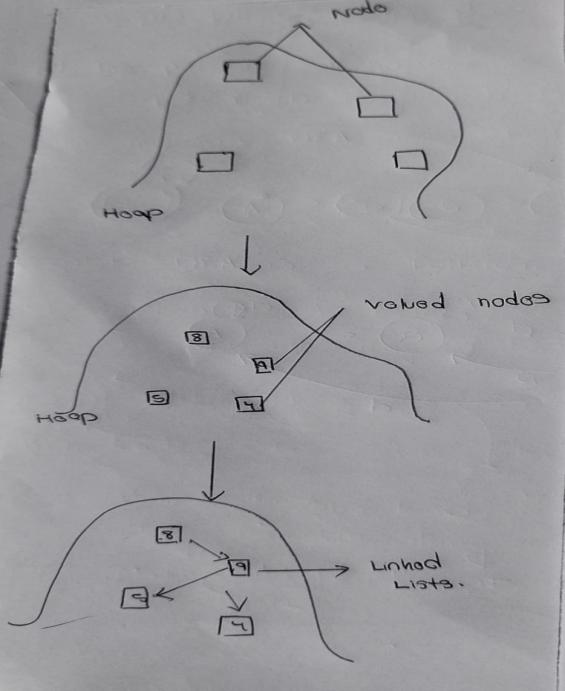
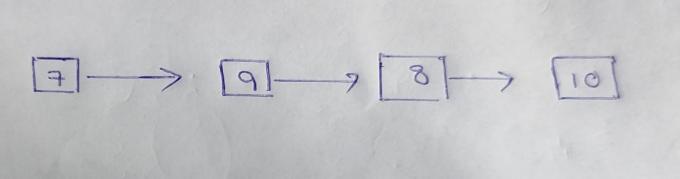
So Linked List is nothing but a data structure. What it is how it is, that is what we are going to unveil in this document.In Java data structures that store multiple data under one variable are:

1. Arrays
2. Array List
3. Linked List

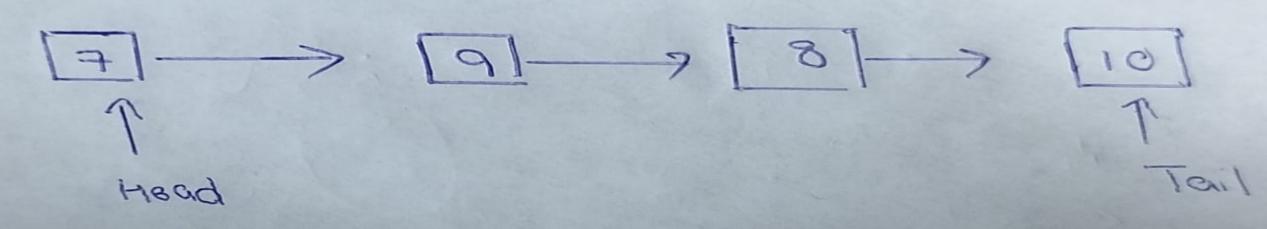
Arrays in Java are discontinuous data structures. Array list again are discontinuous. But they are better than arrays in one aspect that is they are infinite space at their disposal which they utilize as per their requirement. The general implementation is , double the size->Copy the last-> Add the new.

After that comes the great Linked Lists. They are of 3 types. Single, Double And Cyclic Linked List. The general representation of Linked List is as follows.

That how Linked List is made and works. Now we discuss how do we achieve this later. Now as observable Linked List are not continuous memory allocation. They also do not care about memory size and limits are they are constructed out singular valued nodes joined to one another via pointers(not the C++ one). In general, a Linked List looks like this.



Now there are two important pointers that are involved here. The head, and the tail(It might be absent in some case). Head and Tail are not integer but object that store nodes in them(Later Code Will Clarify). It kind of looks like this.

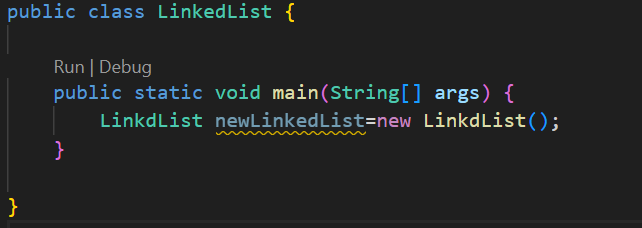


We have discussed a lot about nodes. We should dive deep into it. We will do this via code representation(snippets). So be ready. So before we create a node we require a Linked List to which we would add the nodes. For that:

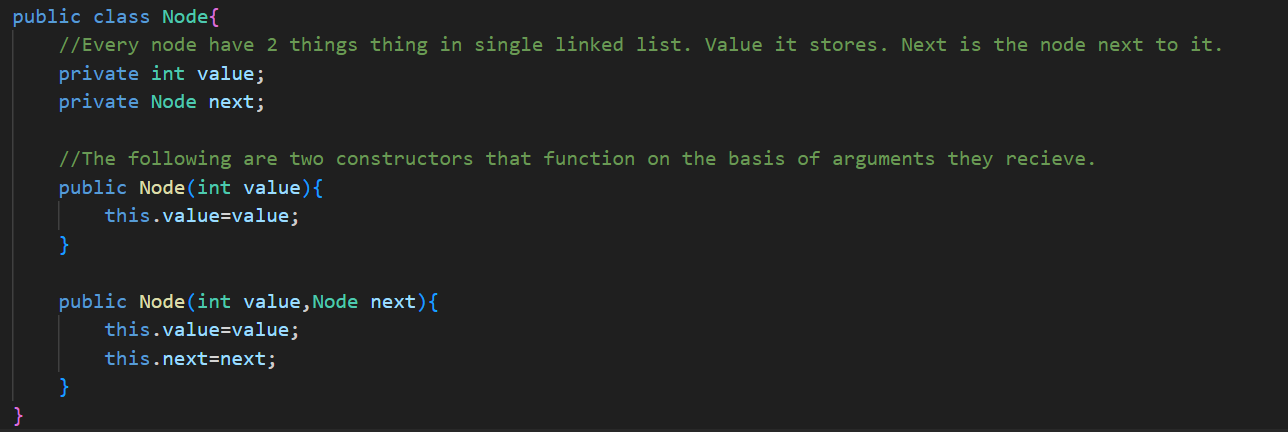
1. We create class Linked List and create its subsequent object in main.

The code snippet show this is as follows.

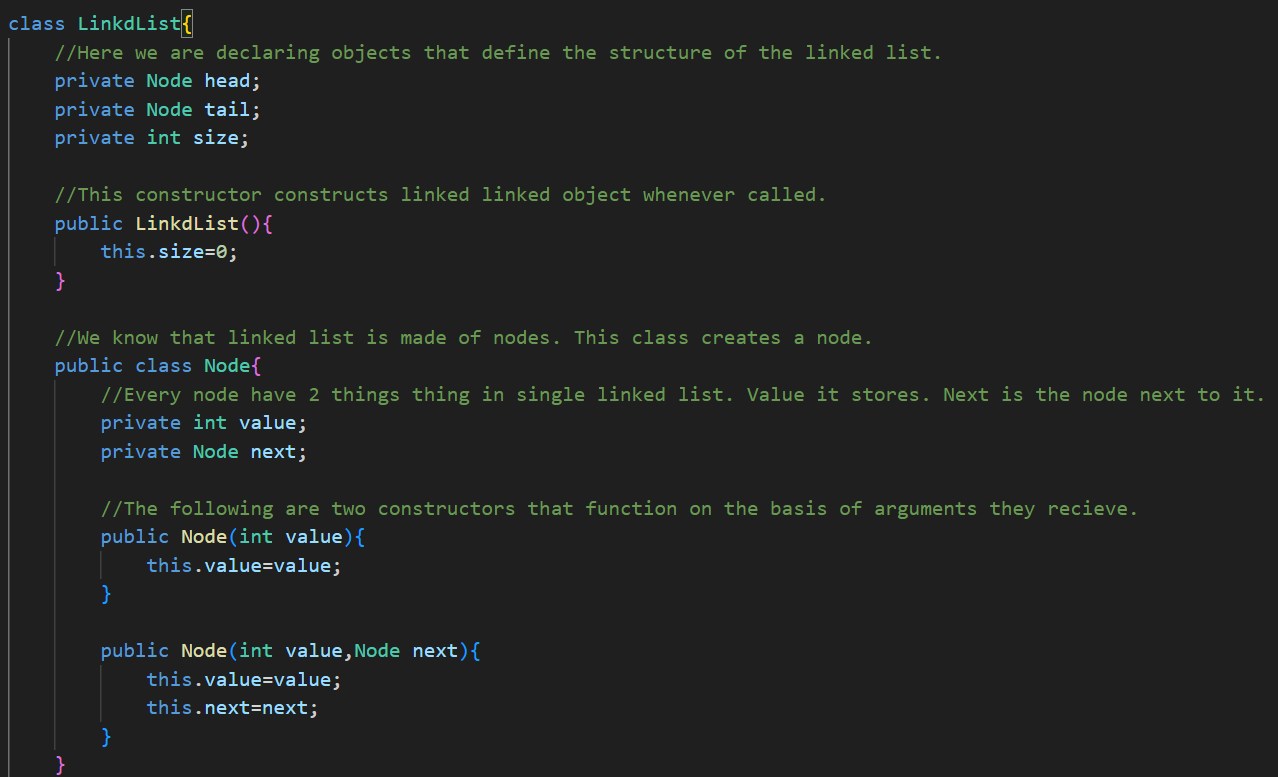




1. Now is the time we add the nodes. Well there is not predefined datatype called nodes. So we need to create one . How do we do that? Well we do it via Classes. Now while discuss classes what are the attribute that this node class should have. Well this is complete related to the type of linked list we are making. Since first we are making single link list we see 2 important attribute that it definitely need to have. 1)Value. 2)Node Next To It. Why? Because that’s the definition of single linked list. The node in it know those 2 things only and in a single linked list we can transverse in one direction only. So yeah nodes? What the code for declaring it?



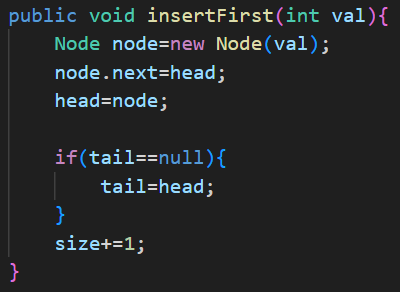
1. So yeah that’s how we create the node of the Single Linked List. Next up is two more important attributes that the class just need to have in order to function properly. Remember HEAD and TAIL. Yeah we gotta declare them too. After their declarence the skeleton code for the class looks like.



Now we move ahead. We now look onto various functions that we can carry in reference to linked lists.

1. **Inserting Element At The Start**

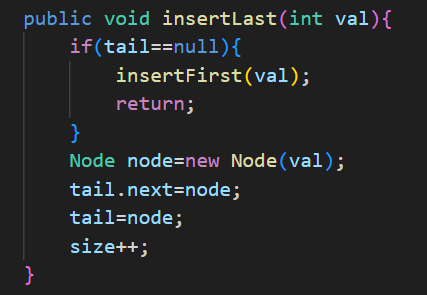
Functional Code is as Follows.

The logic involved is quite simple. Head points to the start of the linked list. So we first create a new node(line 1), the make the head point to this note.

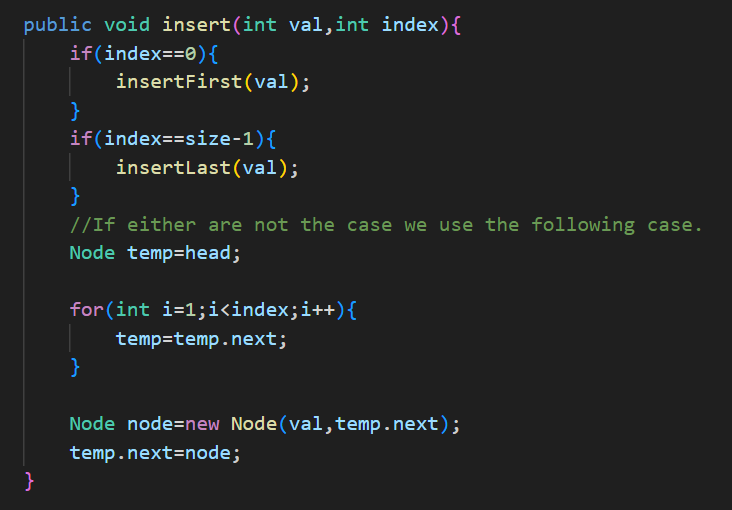
But before we make the head start pointing to new node, we need to move the current head node ahead(line 2) and then make head point to new node(line 3). Another aspect of this is tail variable. This code block is kicked only when length of the LL is 1, so that both the head and tail point to the same node.

1. **Inserting New Element In The Last Index**

The code is as follows:

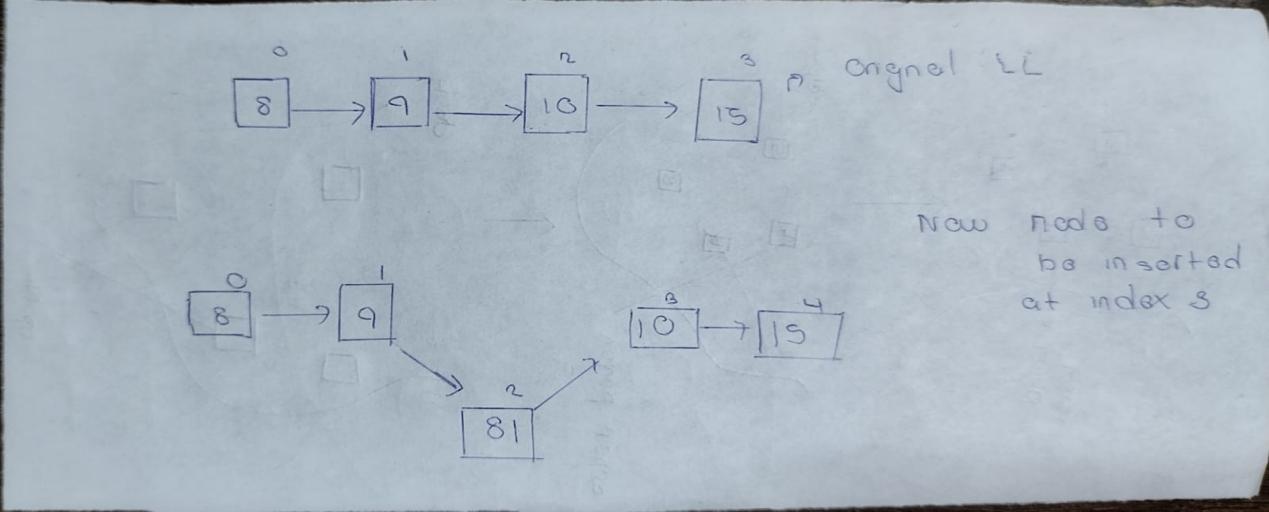
Simple logic. Create new node(Line 5). Place this node next to tail(Line 6). (Line 1,2,3,4)Now make tail point to the newly added node. Of course if the tail is null, this means we are added first element to this LL hence insertFirst() algo would not be bad idea.

1. **Inserting New Element at Any Index Position**

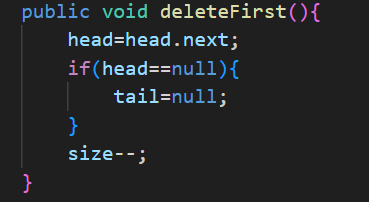
Block 1 and Block 2 can be understood easily.

In general the logic is quite simple.

1. Create a node with the value we inserted.
2. Then now transverse upto node that lies just before the index we want to insert the new node at.
3. Use the node.next command to assign the new node we want to add at position next of node to node at (index-1) hence putting the new node at index.
4. Now reference the node previously at index to node newly inserted at the index, using node.next.



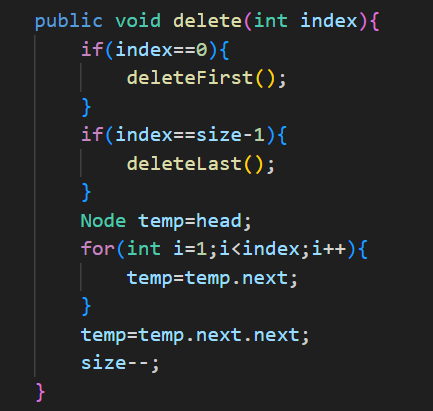
1. **Deleting The First Node**

Here logic is quite simple. We want to remove the first node right? We already know that 2 objects define the structure of the SLL, Head AND Tail. If we move the Head pointer from first element to the second, our SLL starts from 1st index(now 0 in new case) and original Head node is forgotten.

1. **Deleting Element from the last**

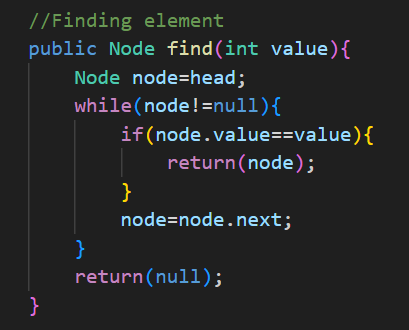
The general logic goes as follows. We assume we only have the head pointer. Them the logic that we involve here is that, we start from head, transverse upto the node that lies just before the before the node pointed by the tail. Then we make this node(node before the last node) pointed by the tail, ignoring the last node in the process hence deleting it.

1. **Deleting Element from any point in the SLL**

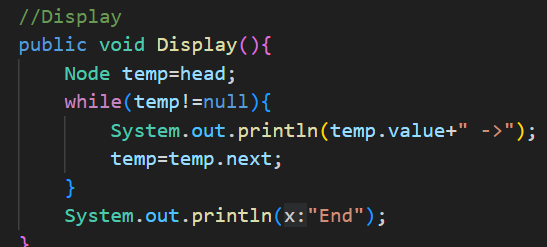


The first two code blocks are self explanatory. Now we move onto the main thing. The logic is transverse upto the node at index just before the one we want to delete. After that when we reach there simply use node.next.next to ignore the node at the middle completely(hence deleting it in the process) and point at the node just after that.

1. **Find Node’s Index Using Value**

Nothing to explain here is just simple linear search algorithm.

1. **Displaying A Linked List**

Again simple transverse and printing while being on the way of doing that.