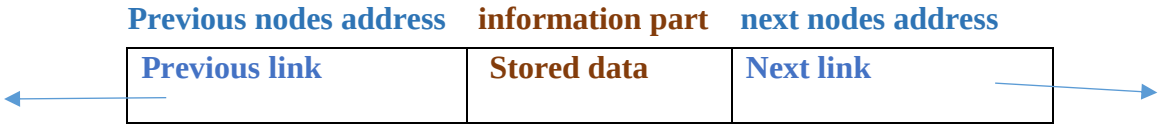
INTRODUCTION TO DOUBLY LINKED LIST

Definition:

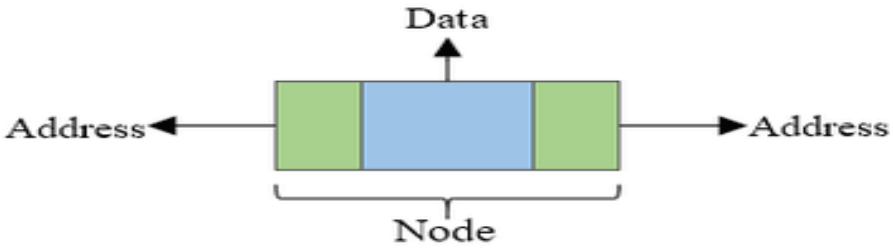
It is a dynamic linear data structure consisting of a linear list or chain of data items called nodes, where each node of the list has information part and two link parts, one link refers to the successor node and another link refers to the predecessor node.

➔ **A node in double linked list contains 3 parts:**

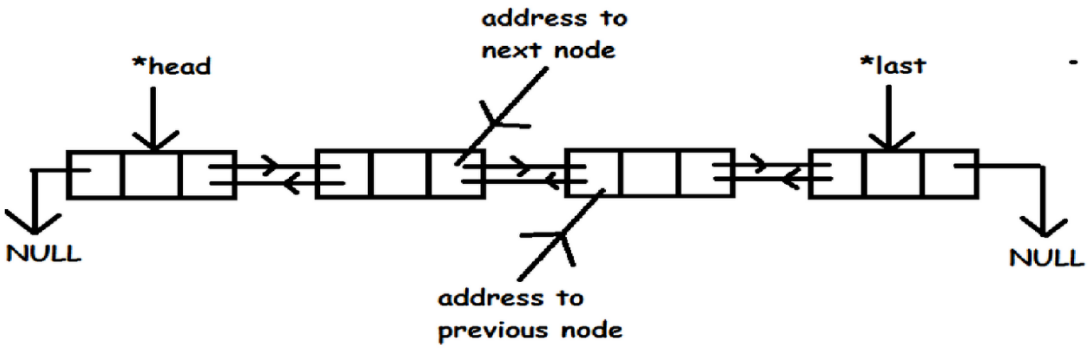
1. **Information part**
2. **Next link part , which stores address of successor node**
3. **Previous link part , which stores address of predecessor node**



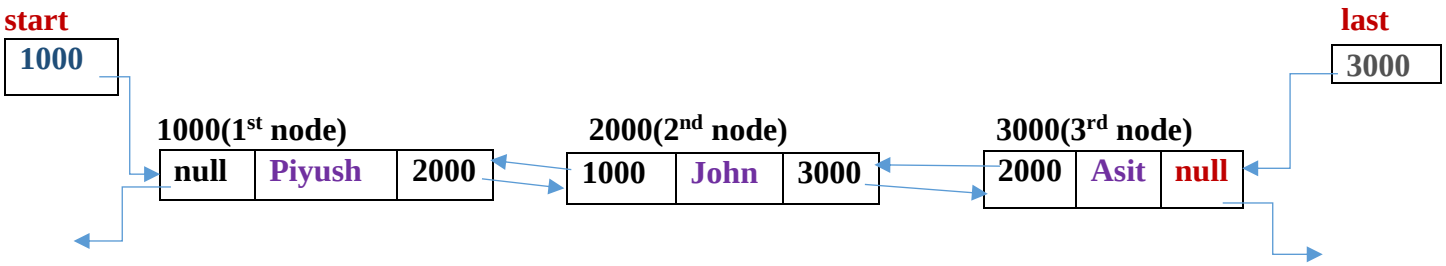
/*Structure of Node in Doubly linked list*/



(Structure of a node in doubly linked list)



(Structure of a doubly linked list)



The above linked list contains name of persons in information part of each node.

A doubly linked list is always associated with two node type reference variables as shown in the above figure.

1. *START or HEAD*, always holds address of first or front node of the list.

2. *LAST or TAIL*, always holds address of last or tail node of the list.

- ➔ **A doubly linked list can be visited or traversed in both directions i.e. in forward as well as in backward directions.**
- ➔ **While traversing in forward direction, start from the first or front node and using the *next link* part of each node we visit the successor nodes.**
- ➔ **While traversing in backward direction, start from the last or tail node and using the *previous link* part of each node we visit the predecessor nodes.**
- ➔ **The *next link* part of last node holds *'null'* value to indicate end or last node of the list.**
- ➔ **The *previous link* part of first node holds *'null'* value to indicate first node or beginning of the list.**

Advantage of doubly linked list over single linked list:

- ➔ **A single linked list can only be visited in one direction because each node only holds the address of successor nodes.**
- ➔ **Whereas a doubly linked list can be visited in both directions because each node holds the address of both successor and predecessor nodes.**

Operation performed on doubly linked lists:

1. **Create a doubly linked list**
2. **Traverse**
3. **Insertion**
4. **Deletion**
5. **Counting**
6. **Update**
7. **Search a node**
8. **Sorting the list**

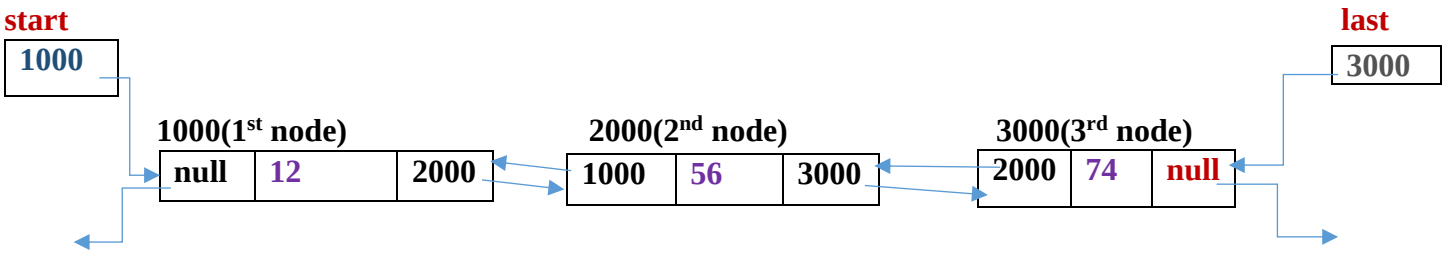
Implementation of doubly linked list:

To create a doubly linked list where each node will hold an integer value declare a node type class as follows:

```
class DLLNODE
{
    int info;
    DLLNODE nextlink;
    DLLNODE prevlink;
}
        Or
class NODE
{
    int info;
```

```
    DLLNODE nextlink;
    DLLNODE prevlink;
}
```

A doubly linked list which uses the above class declarations to created nodes is as shown in the following figure:



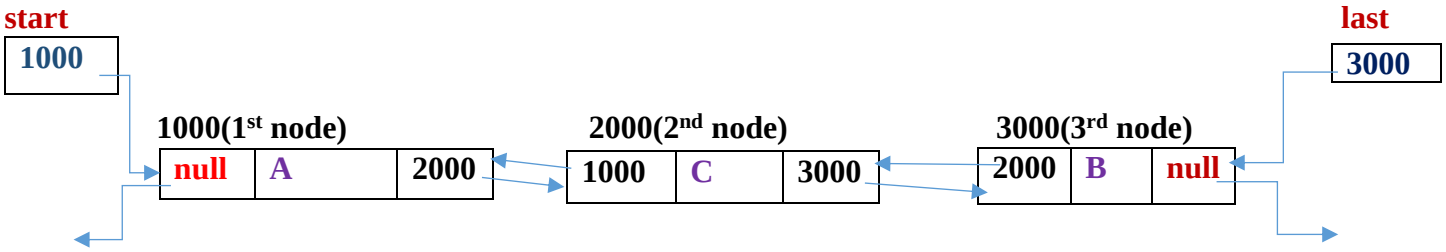
To create a doubly linked list where each node will hold grade obtained by students, you can declare a node type class as follows:

```
class NODE_GRADE
{
    char  grade;
    NODE_GRADE nextlink;
    NODE_GRADE prevlink;
}
```

Or

```
class DLLNODE
{
    char  grade;
    DLLNODE nextlink;
    DLLNODE prevlink;
}
```

A doubly linked list which uses the above class declarations to created nodes is as shown in the following figure:

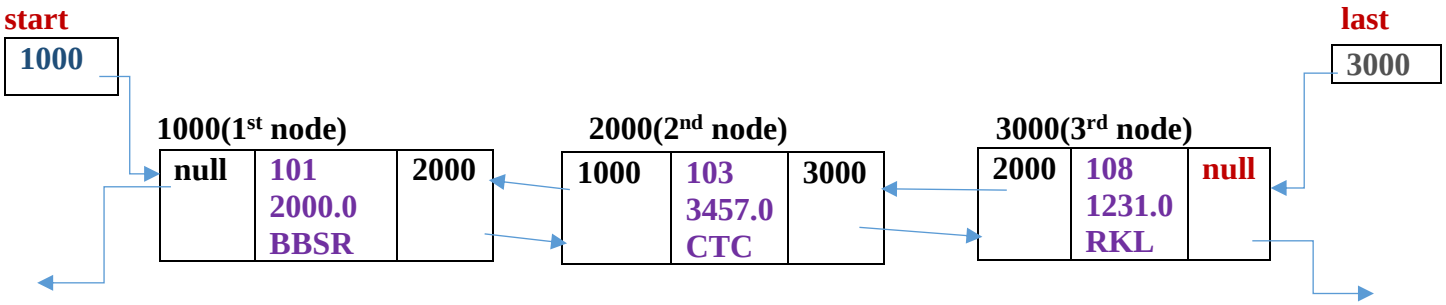


To create a doubly linked list where each node will hold information of bank accounts (Account no, balance, branch name), you can declare a node type class as follows:

```
class account_node
{   int account_no;
    float balance;
    String branch_name;

    account_node nextlink;
    account_node prevlink;
}
```

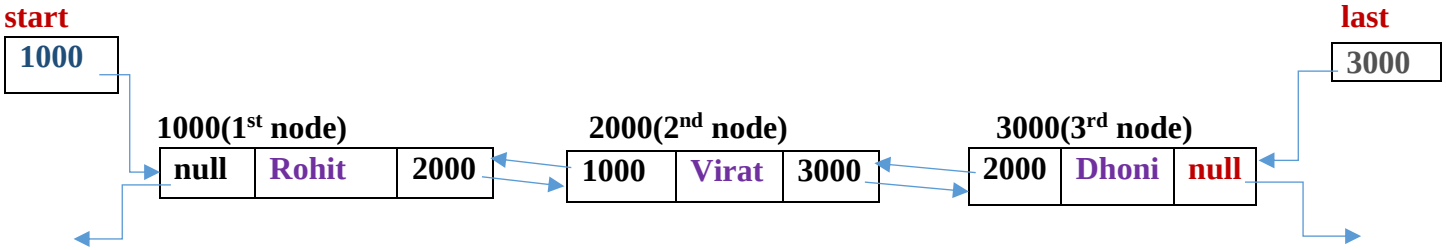
A doubly linked list which uses the above class declarations to created nodes is as shown in the following figure:



Some of the java statements which are frequently used in programming on doubly linked lists:

Example: Let's consider the following doubly linked list and the class declaration to create nodes of the list. The following list holds name a person in each node, so the node type class can be declared as follows:

```
class Person_node
{
    String person_name;
    Person_node nextlink;
    Person_node prevlink;
}
```



Initially there is no doubly linked list, so to create an empty doubly linked list write the following java statement:

static Person_node *start*=null; /* 'start' pointer will always refer to first node of the list*/

static Person_node *last*=null; /* 'last' pointer will always refer to last node of the list*/

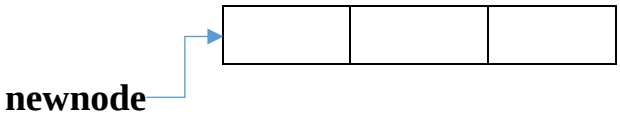
Initial empty doubly linked list created after the above two statements is as shown in following figure:

Just like creating the nodes in single linked list, to create a node of doubly linked list write the following java statement by calling default constructor of *Person_node* class:

Person_node newnode=new Person_node(); /*← creates a new node and assigns the nodes address to the 'Person_node' type reference variable 'newnode' shown in the following figure*/

/* New node created by the above statement*/

Address of the node→ 1000 (new node)



/*To store data in information part of the above node write following java statement:*/

newnode.person_name="dhoni"; /*assigns "dhoni" to person_name in the info part*/

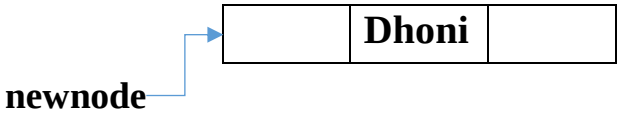
/*To accept the data from keyboard write the following java statement:*/

System.out.println("Enter name of the person to be stored in the new node");

newnode.person_name=sc.next(); /*← stores the entered name in person_name */

/*after entering data in information part the node is as follows:*/

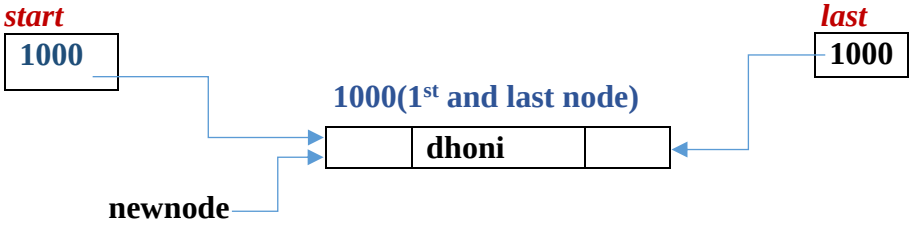
Address of the node→ 1000 (new node)



/*To insert or attach the above node into the following empty linked list we have created*/

Write:

start=last=newnode; /*←stores the new nodes address both in 'start' and 'last' pointer because the new node would be the first as well as last node of the list as shown in following figure:*/



/*Then store null value both in previous and next link part of the above node because it is both first and last node of the list*/

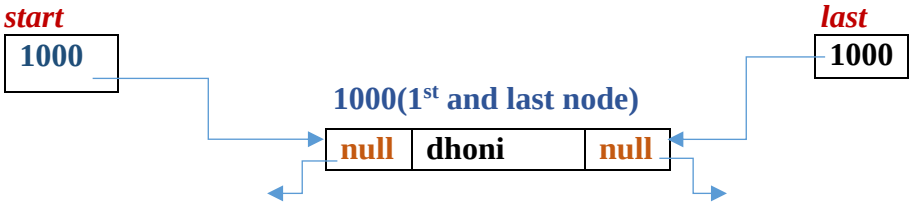
newnode.prevlink=newnode.nextlink=null; /*stores null in link parts*/

or

You can also write:

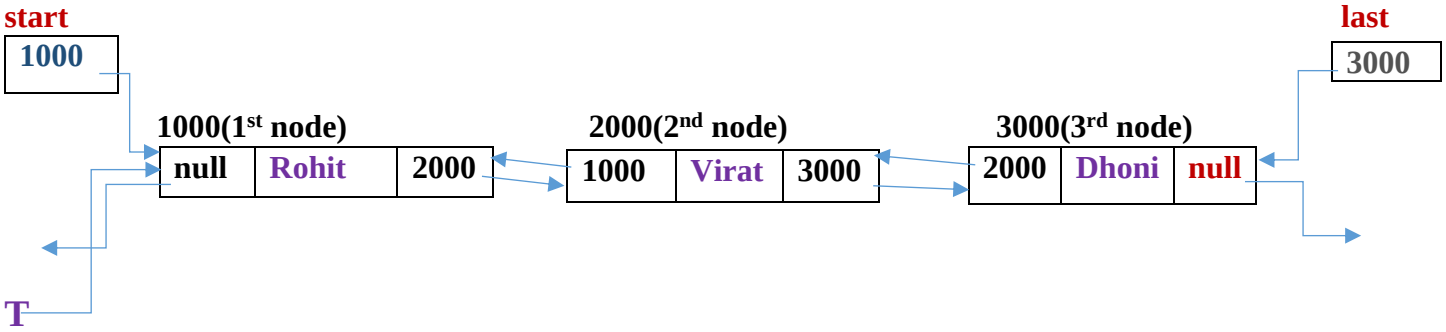
start.prevlink=start.nextlink=null; /* because 'start' also refers to same node*/

/*so the final doubly lined list is as shown in the following figure:*/



/*JAVA statements to move from one node to next node or previous node*/

/*Consider the following doubly linked list:*/



Person_node T=start; /*←T refers to first node as shown in the above figure:*/

/*don't modify the value of 'start' or 'last', use a temporary Person_node type reference variable like T to move from one node to another*/

/*Now T is referring to 1st node, so if you write following java statements:*/

System.out.println("1st persons name is: " + T.person_name); /*← prints Rohit*/

System.out.println("1st persons name is: " + T.nextlink.person_name); /*← prints Virat*/

/*T.nextlink→refers to 2nd nodes address which is the successor of 1st node, so T.nextlink.person_name→ info part of 2nd node i.e. person_name value. T.prevlink→ null , because there is no node before the first node, so writing T.prevlink.person_name will give you nothing or incorrect*/

To move to 2nd node write :

T=T.nextlink; /*← assigns 2nd nodes address to T, T gets updated to 2nd node*/

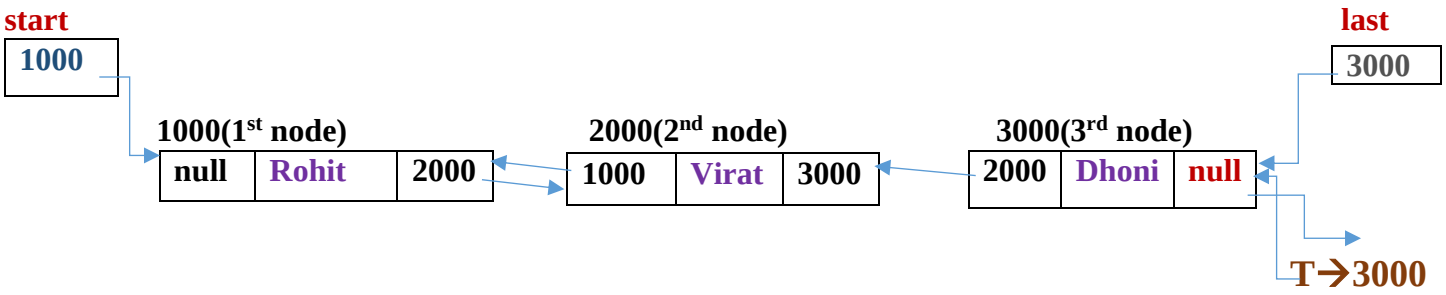
Therefore T moves to 2nd node. Now if you write the following statements:*/

System.out.println("1st persons name is: " + T.person_name); /*← prints Virat*/

System.out.println("1st persons name is: " + T.prevlink.person_name); /*← prints Rohit*/

/* if you want to visit the nodes of the list in backward direction i.e. from last to the first node then initialize T to last node by 'last' reference variable*/

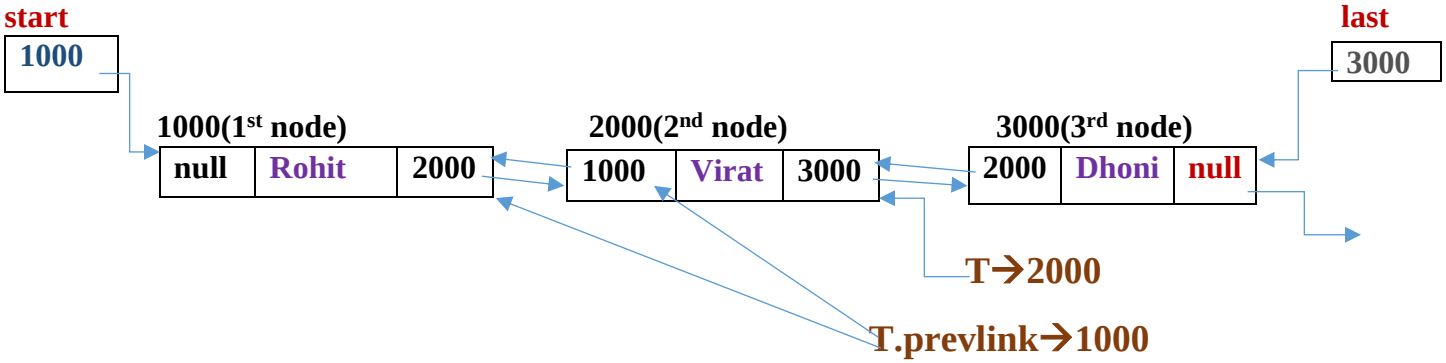
Person_node T=last; /*← T refers to last node of the list as shown in following figure*/



So, if you write:

System.out.println("1st persons name is: " + T.person_name); /*← prints Dhoni*/

T=T.prevlink;/*← T moves to 2nd node i.e. previous of the last node in backward direction*/



System.out.println("1st persons name is: " + T.person_name); /*← prints Virat*/

System.out.println("1st persons name is: " + T.prevlink.person_name); /*← prints Rohit*/

T.prevlink.prevlink→ Gives you content of 1st nodes previous link part→ null