START
Node head;

Node tail;

class Node

String data;

Node prev;

Node next;

//Node tail = null;

Node(String d)

data = d;

Node head, tail = null;

addNode(String data)

Node newNode = new Node(data);

if (head == null)

head = tail = newNode;

head.prev = null;

tail.next = null;

else

tail.next = newNode;

newNode.prev = tail;

tail = newNode;

tail.next = null;

```
// insert node at the front
public void addinFront(String data)
// allocate memory for newNode and assign data to newNode
Node newNode = new Node(data);

// make newNode as a head
newNode.next = head;

// assign null to prev of newNode
newNode.prev = null;

// previous of head (now head is the second node) is newNode
if (head != null)
head.prev = newNode;

// head points to newNode
head = newNode;

// insert a node after a specific node
addAfter(Node prev_node, String data)

// check if previous node is null
if (prev_node == null) {
System.out.println("previous node cannot be null");
```

```
return;

// allocate memory for newNode and assign data to newNode
Node new_node = new Node(data);

// set next of newNode to next of prev node
new_node.next = prev_node.next;

// set next of prev node to newNode
prev_node.next = new_node;

// set prev of newNode to the previous node
new_node.prev = prev_node;

// set prev of newNode's next to newNode
if (new_node.next != null)
new_node.next.prev = new_node;

// insert a newNode at the end of the list
void insertEnd(String data) {
// allocate memory for newNode and assign data to newNode
Node new_node = new Node(data);

// store the head node temporarily (for later use)
Node temp = head;

// assign null to next of newNode
```

```
new_node.next = null;

// if the linked list is empty, make the newNode as head node
if (head == null)
new_node.prev = null;
head = new_node;
return;

// if the linked list is not empty, traverse to the end of the linked list
while (temp.next != null)
temp = temp.next;

// assign next of the last node (temp) to newNode
temp.next = new_node;

// assign prev of newNode to temp
new_node.prev = temp;

deletePos(int n)
if(head == null)
return;
else
Node current = head;
int pos =n;

for(int i = 1; i < pos; i++){
current = current.next;
```

```
if(current == head)
head = current.next;

else if(current == tail)
tail = tail.prev;

else
current.prev.next = current.next;
current.next.prev = current.prev;

current = null;

// print the doubly linked list
public void printlist(Node node)
Node last = null;
while (node != null)
System.out.print(node.data + "-->");
last = node;
node = node.next;

System.out.println();

searchNode(String value)
int i = 1;
boolean flag = false;
//Node current will point to head
Node current = head;
```

```
//Checks whether the list is empty

if (head == null)

System.out.println("List is empty");

return;


int fnd = 0;

while (current != null)

//Compare value to be searched with each node in the list


if ((current.data.equals(value)) == true)
Display("the same ");

Display("Node is present in the list at the position : " + i);

break;

else


Display ("Node is not present in the list")

current = current.next;

i++;

        printReverse(Node head_ref)

Node tail = head_ref;


// Traversing till tail of the linked list

while (tail.next != null)


tail = tail.next;


// Traversing linked list from tail

// and printing the node.data
```

```
while (tail != head_ref)

Display ( tail.data + "--> ");

tail = tail.prev;

Display ( tail.data );


DoublyLinkedList Playlist = new DoublyLinkedList();


Playlist.printlist(Playlist.head);


int b=0;

while(b<10){




Display ("----------Music Playing App ver. 2.5.1----------\n  \n" +

"1. add song at the front.\n2. Add song at the End. \n3. delete song. \n4. Print (play) list \n5. Play in reverse \n6. Search song\n7. Close app \n ");

int option = sc.nextInt();

switch (option)


// Case add song infront

case 1:

Display ("How many songs do you want to add at the front? ");

int cse1 = sc.nextInt();


for (int i = 0; i < cse1; i++)

Display ("enter name of song to add integer");

String songadd = sc.next();

Playlist.addinFront(songadd);
```

```
break;

// Case add song at the end
case 2:
Display ("How many songs do you want to add at the End? ");
int cse2 = sc.nextInt();

for (int i = 0; i < cse2; i++)
Display ("enter name of song to add at the end (integer)");
String songadd1 = sc.next();
Playlist.insertEnd(songadd1);



break;

// Case delete song
case 3:
Display ("Enter position of song you want to delete");
int Posdelete= sc.nextInt();
Playlist.deletePos(Posdelete);

break;

// Case print list
case 4:
Display ("1. Repeat on? \n2. Repeat off \n  ");
```

```
int rpt= sc.nextInt();
if(rpt==1){

//doubly_ll.printrepeat(doubly_ll.head);
int kk=0;
System.out.println("The printed list is: ");
while(kk<10)

Playlist.printlist(Playlist.head);
/*System.out.println("Stop 1. yes / 2. no");
int stp = sc.nextInt();
if(stp==1){
break;
else
continue;*/

else if(rpt==2)
Display ("The printed list is: ");
Playlist.printlist(Playlist.head);
break;

//print in reverse
case 5:
Display (" List in reverse order is ");
Playlist.printReverse(Playlist.head);
break;
```

```
// search for a song

case 6:

Display (" Enter song you want to search");

String srch = sc.next();

Playlist.searchNode(srch);

break;


// close app

case 7:

b=10;

break;


// Default case

default:

Display ("Error: input type not permitted");


Display ("The printed list is: ");

Playlist.printlist(Playlist.head );
END
```

START