CIRCULAR LINKED LIST

1. SINGLE LINKED LIST

- **INSERTION**

1.Insert in empty list

```java
class Node {
    int data;
    Node next;

    Node(int value) {
        data = value;
        next = null;
    }
}

class LinkedList {
    // Function to insert a node into an empty
    // circular singly linked list
    static Node insertInEmptyList(Node last, int data) {
        if (last != null) return last;
        Node newNode = new Node(data); // Create a new node
        newNode.next = newNode; // Point newNode to itself
        last = newNode;// Update last to point to the new node
        return last;
    }
    // Function to print the list
    static void printList(Node last) {
        if (last == null) return;
        Node head = last.next; // Start from the head node
        while (true) {
            System.out.print(head.data + " ");
            head = head.next;
            if (head == last.next) break;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Node last = null;
        last = insertInEmptyList(last, 1);
        printList(last);
    }
}
```

```
OUTPUT:
1
```

2.Insert at Frist

```
class Node {
    int data;
    Node next;
    Node(int value){
        data = value;
        next = null;
    }
}

public class Linkedlist {
    // Function to insert a node at the beginning of the
    // circular linked list
    public static Node insertAtBeginning(Node last,int value){
        Node newNode = new Node(value);
        if (last == null) {  // If the list is empty, make the new
node point to itself and set it as last
            newNode.next = newNode;
            return newNode;
        }
        newNode.next = last.next;// Insert the new node at the
beginning
        last.next = newNode;
        return last;
    }
    // Function to print the circular linked list
    public static void printList(Node last){
        if (last == null)
            return;
        Node head = last.next;
        while (true) {
            System.out.print(head.data + " ");
            head = head.next;
            if (head == last.next) break;
        }
        System.out.println();
    }

    public static void main(String[] args){
        Node first = new Node(2);
        first.next = new Node(3);
        first.next.next = new Node(4);
        Node last = first.next.next;
        last.next = first;
        last = insertAtBeginning(last, 5);
        printList(last);
    }
}
```

```
OUTPUT:
5 2 3 4
```

3.Insert at last

```
class Node{
  int data;
  Node next;
  public Node(int data){
    this.data=data;
    this.next=null;
  }
}
class Linkedlist{
  public static Node insert(Node last,int data){
   Node newnode= new Node(data);
   if(last==null){
   // If the list is empty, initialize it with the
   // new node
    last=newnode;
    newnode.next=newnode;
   }
   else{
             // Insert new node after the current tail and
            // update the tail pointer
    newnode.next=last.next;
    last.next=newnode;
    last=newnode;
   }
   return last;
  }
  public static void trv(Node last){
    if(last==null){
      return;
    }
    Node head=last.next;
    while(true){
      System.out.print(head.data+" ");
      head=head.next;
      if(head==last.next){
        break;
      }
      //System.out.println();
    }
  }
  public static void main(String[] args){
    Node frist=new Node(1);
    frist.next=new Node(2);
    frist.next.next=new Node(3);
    Node last=frist.next.next;
    last.next=frist;
    int data=5;
    last=insert(last, data);
    trv(last);
  }
}
OUTPUT:
1 2 3 4 5
```

4.Insert at any position

```java
class Node {
    int data;
    Node next;

    Node(int value){
        data = value;
        next = null;
    }
}

public class GFG {

    // Function to insert a node at a specific position in a
    // circular linked list
    static Node insertAtPosition(Node last, int data,int pos){
        if (last == null) {
            // If the list is empty
            if (pos != 1) {
                System.out.println("Invalid position!");
                return last;
            }
            Node newNode = new Node(data); // Create a new node and make it point to itself
            last = newNode;
            last.next = last;
            return last;
        }
        Node newNode = new Node(data); // Create a new node with the given data
        Node curr = last.next;  // curr will point to head initially
        if (pos == 1) {
            // Insert at the beginning
            newNode.next = curr;
            last.next = newNode;
            return last;
        }
        // Traverse the list to find the insertion point
        for (int i = 1; i < pos - 1; ++i) {
            curr = curr.next;
            // If position is out of bounds
            if (curr == last.next) {
                System.out.println("Invalid position!");
                return last;
            }
        }
        newNode.next = curr.next;// Insert the new node at the desired position
        curr.next = newNode;
        // Update last if the new node is inserted at the end
        if (curr == last)
            last = newNode;
        return last;
    }

    static void printList(Node last){
        if (last == null)
            return;

        Node head = last.next;
        while (true) {
            System.out.print(head.data + " ");
            head = head.next;
            if (head == last.next)
                break;
        }
        System.out.println();
    }

    public static void main(String[] args)
    {
        // Create circular linked list: 2, 3, 4
        Node first = new Node(2);
        first.next = new Node(3);
        first.next.next = new Node(4);
        Node last = first.next.next;
        last.next = first;
        int data = 5, pos = 2;
        last = insertAtPosition(last, data, pos);
        printList(last);
    }
}
OUTPUT:
2 5 3 4
```

- **DELETIONS**

  1.Delete at Frist

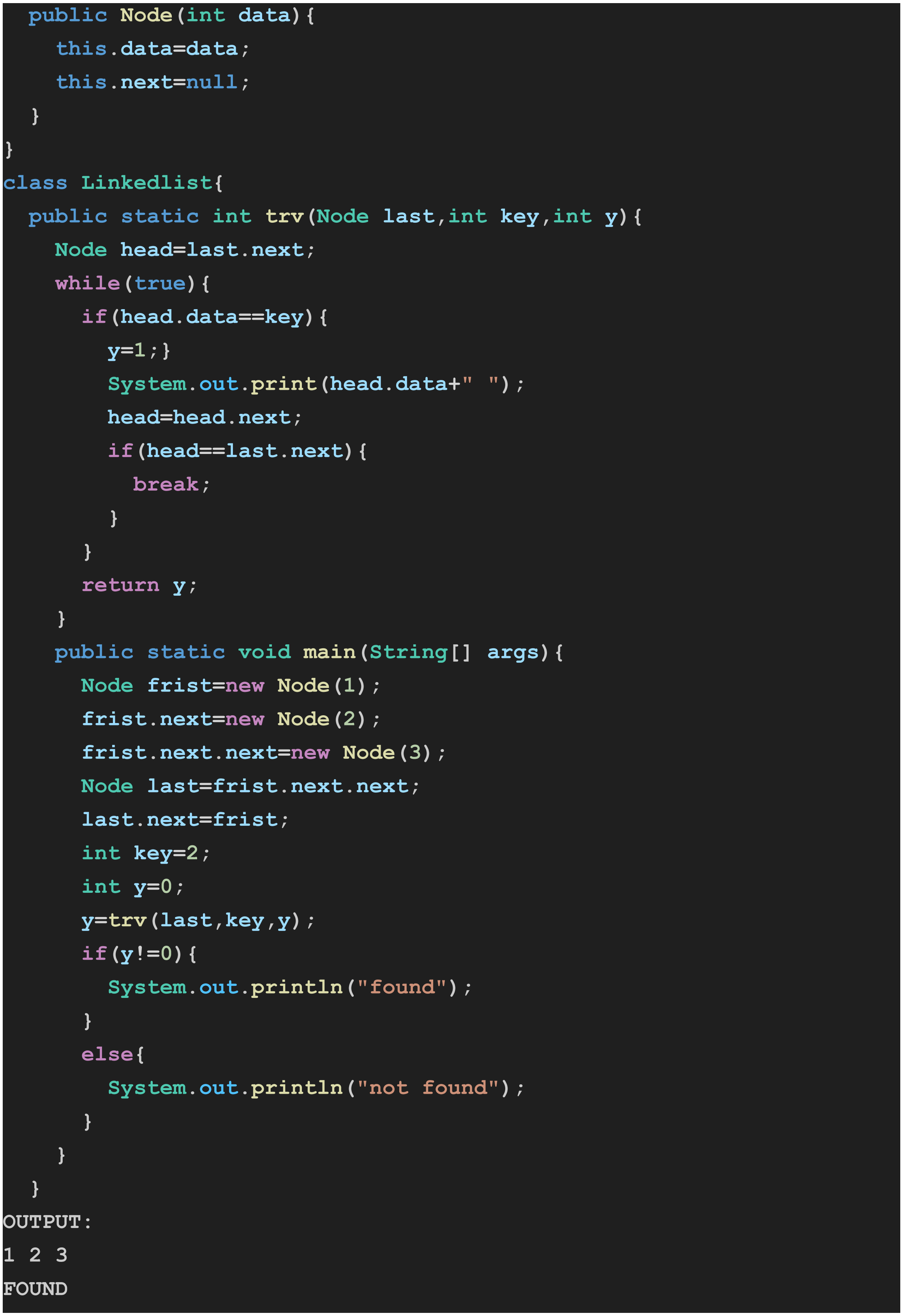
```
 class Node{
    int data;
    Node next;
    public Node(int data){
        this.data=data;
        this.next=null;
    }
}
class Linkedlist{
    public static Node delete(Node last){
        if(last==null){
            System.out.println("list is empty");
            return null;
        }
        Node head=last.next;
        if(head==last){//isf there is no node in ths list
            last=null;
        }
        else{//more than one node in the list
            last.next=head.next;
        }
        return last;
    }
    public static void trv(Node last){
        if(last==null){
            return;
        }
        Node head=last.next;
        while(true){
            System.out.print(head.data+" ");
            head=head.next;
            if(head==last.next){
                break;
            }
        }
    }
    public static void main(String[] args){
        Node frist= new Node(1);
        frist.next=new Node(2);
        frist.next.next=new Node(3);
        Node last=frist.next.next;
        last.next=frist;
        last=delete(last);
        trv(last);
    }
}
OUTPUT:
2 3
```

3.Delete at End

```
class Node{
    int data;
    Node next;
    public Node(int data){
        this.data=data;
        this.next=null;
    }
}
class Linkedlist{
    public static Node delete(Node last){
      if(last==null){
        System.out.println("list is empty");
        return null;
      }
      Node head=last.next;
      if(head==last){//if only one node in list
        last=null;
        return last;
      }
      Node curr=head;
      while(curr.next!=last){ // Traverse the list to find the
second last node
        curr=curr.next;
      }
      curr.next=head; // Update the second last node's next pointer
to point to head
      last=curr;
      return last;
    }
    public static void trv(Node last){
        if(last==null){
            return;
        }
        Node head=last.next;
        while(true){
            System.out.print(head.data+" ");
            head=head.next;
            if(head==last.next){
                break;
            }
        }
    }
    public static void main(String[] args){
        Node frist= new Node(1);
        frist.next=new Node(2);
        frist.next.next=new Node(3);
        Node last=frist.next.next;
        last.next=frist;
        last=delete(last);
        trv(last);
    }
}
OUTPUT:
1 2
```

- **SEARCHING ELEMENT**

```
class Node{
  int data;
  Node next;
```



```
  public Node(int data){
    this.data=data;
    this.next=null;
  }
}
class Linkedlist{
  public static int trv(Node last,int key,int y){
    Node head=last.next;
    while(true){
      if(head.data==key){
        y=1;}
        System.out.print(head.data+" ");
        head=head.next;
        if(head==last.next){
          break;
        }
      }
      return y;
    }
    public static void main(String[] args){
      Node frist=new Node(1);
      frist.next=new Node(2);
      frist.next.next=new Node(3);
      Node last=frist.next.next;
      last.next=frist;
      int key=2;
      int y=0;
      y=trv(last,key,y);
      if(y!=0){
        System.out.println("found");
      }
      else{
        System.out.println("not found");
      }
    }
  }
OUTPUT:
1 2 3
FOUND
```