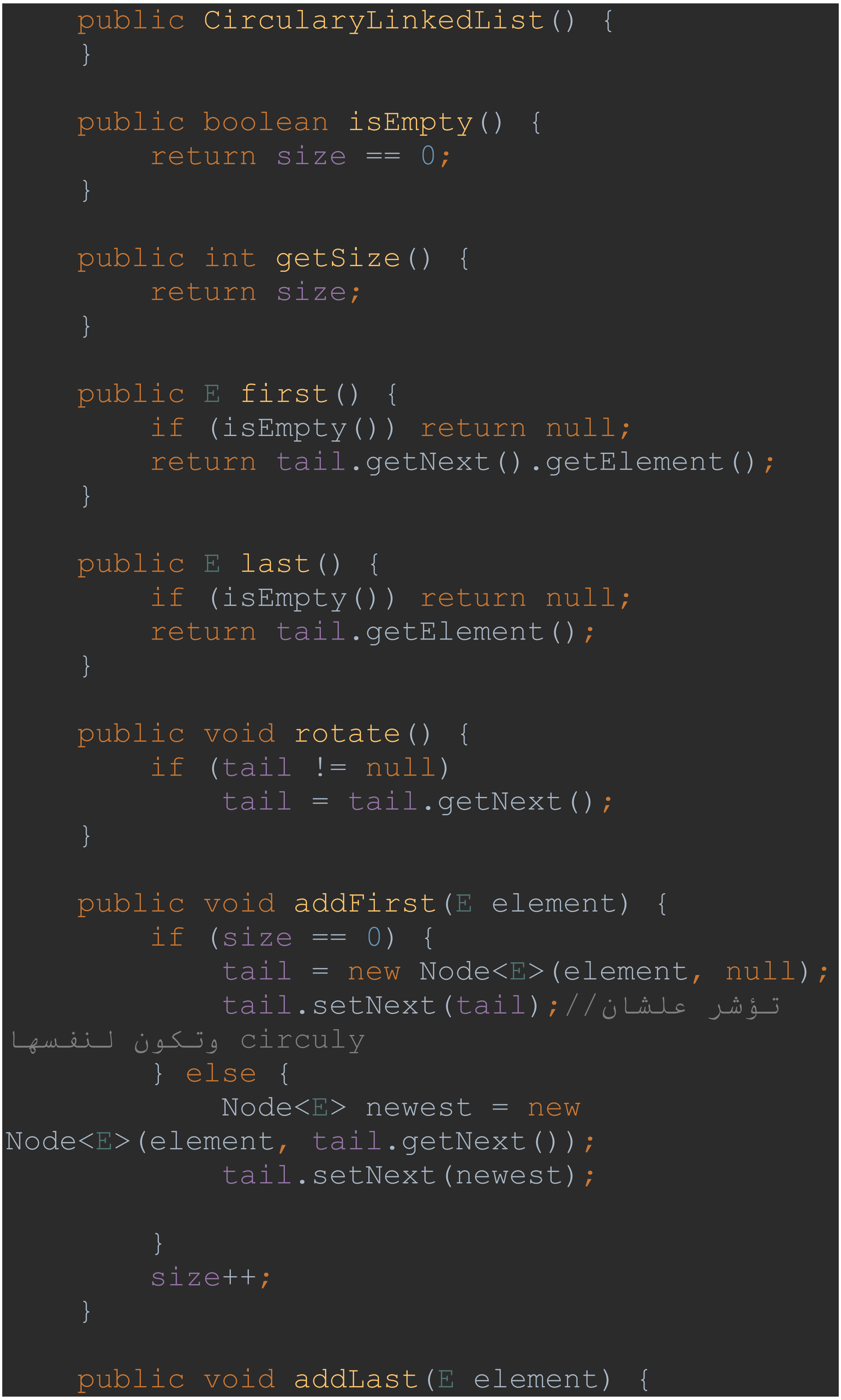
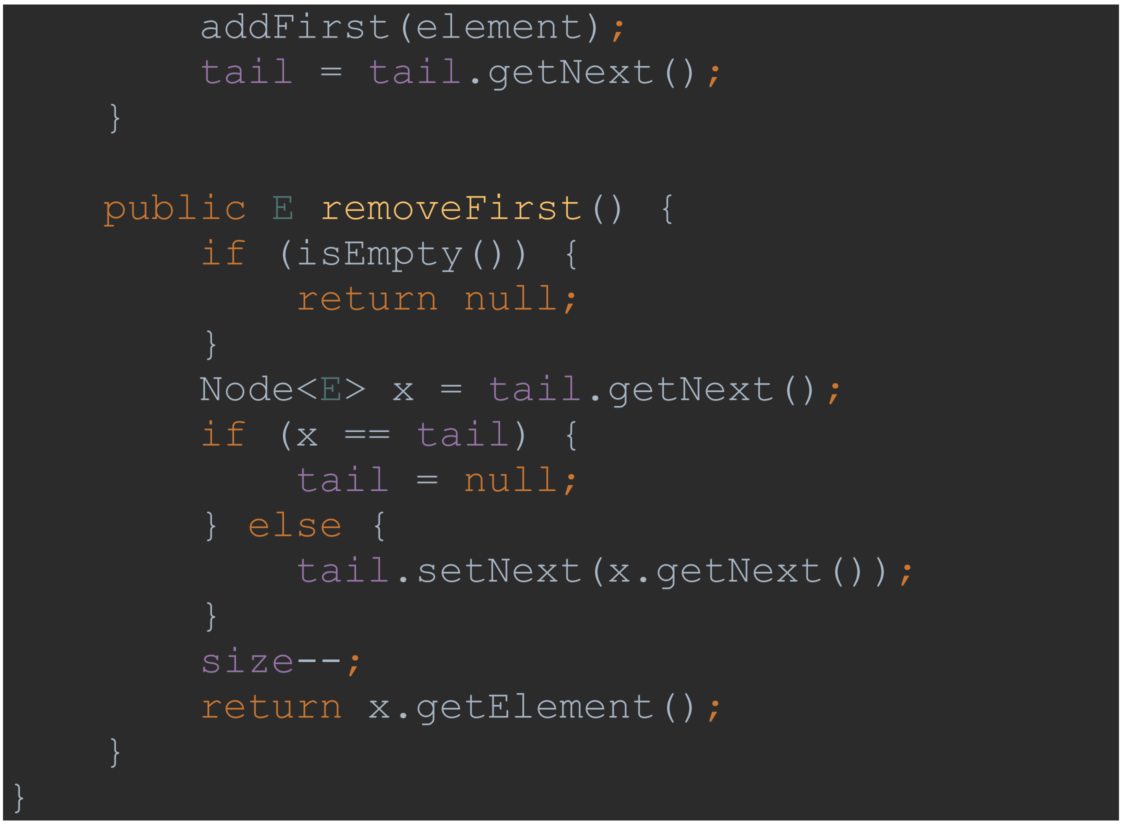
Lab6:

1. **Round robin scheduling**
2. **Circularly linked list**
3. **Print elements of CircularlyLinkedList**
4. **Doubly linked list**
5. **Print elements of DoublyLinkedList in reverse order**

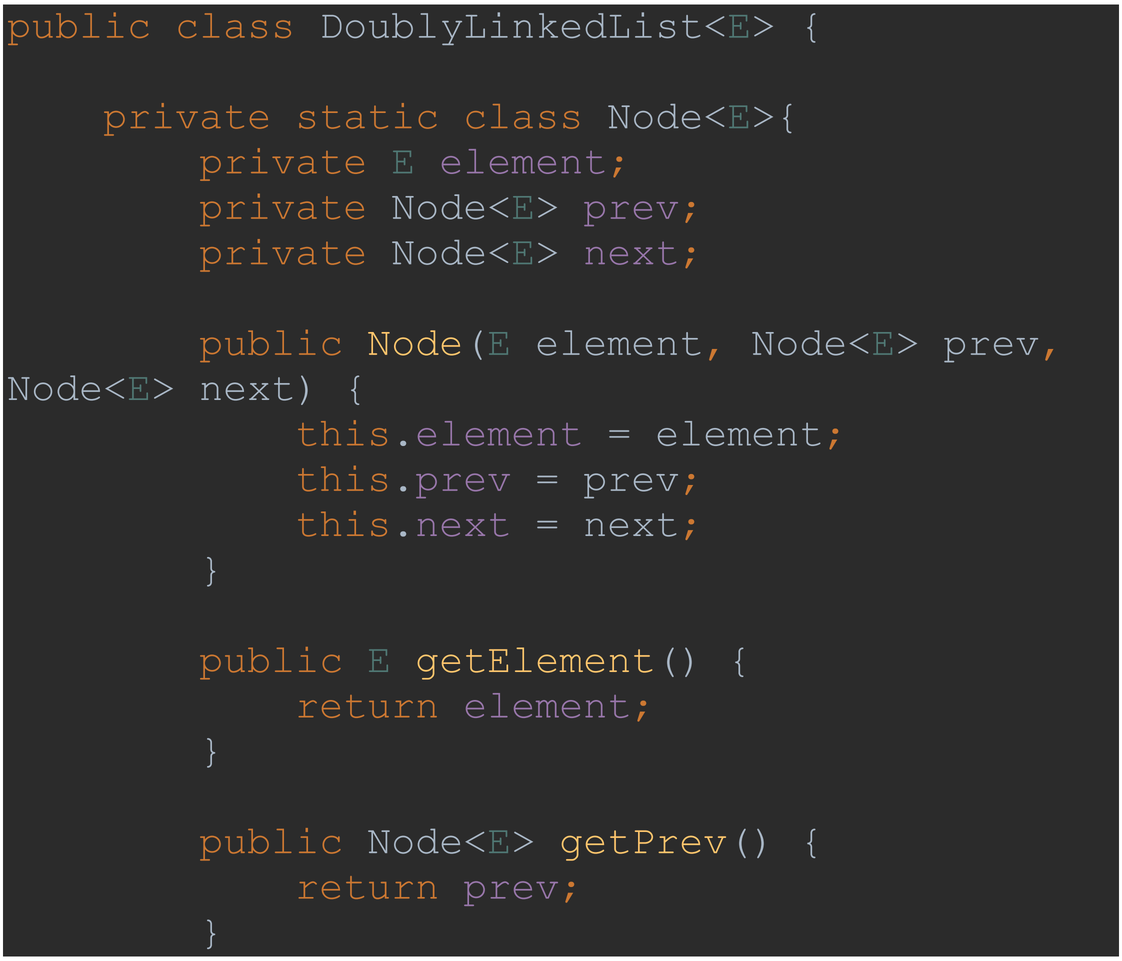
```
public class CircularyLinkedList<E> {
    private static class Node<E> {
        private E element;
        private Node<E> next;

        public Node(E element, Node<E> next) {
            this.element = element;
            this.next = next;
        }

        public E getElement() {
            return element;
        }

        public void setElement(E element) {
            this.element = element;
        }

        public Node<E> getNext() {
            return next;
        }

        public void setNext(Node<E> next) {
            this.next = next;
        }
    }

    private Node<E> tail = null;
    private int size = 0;
```



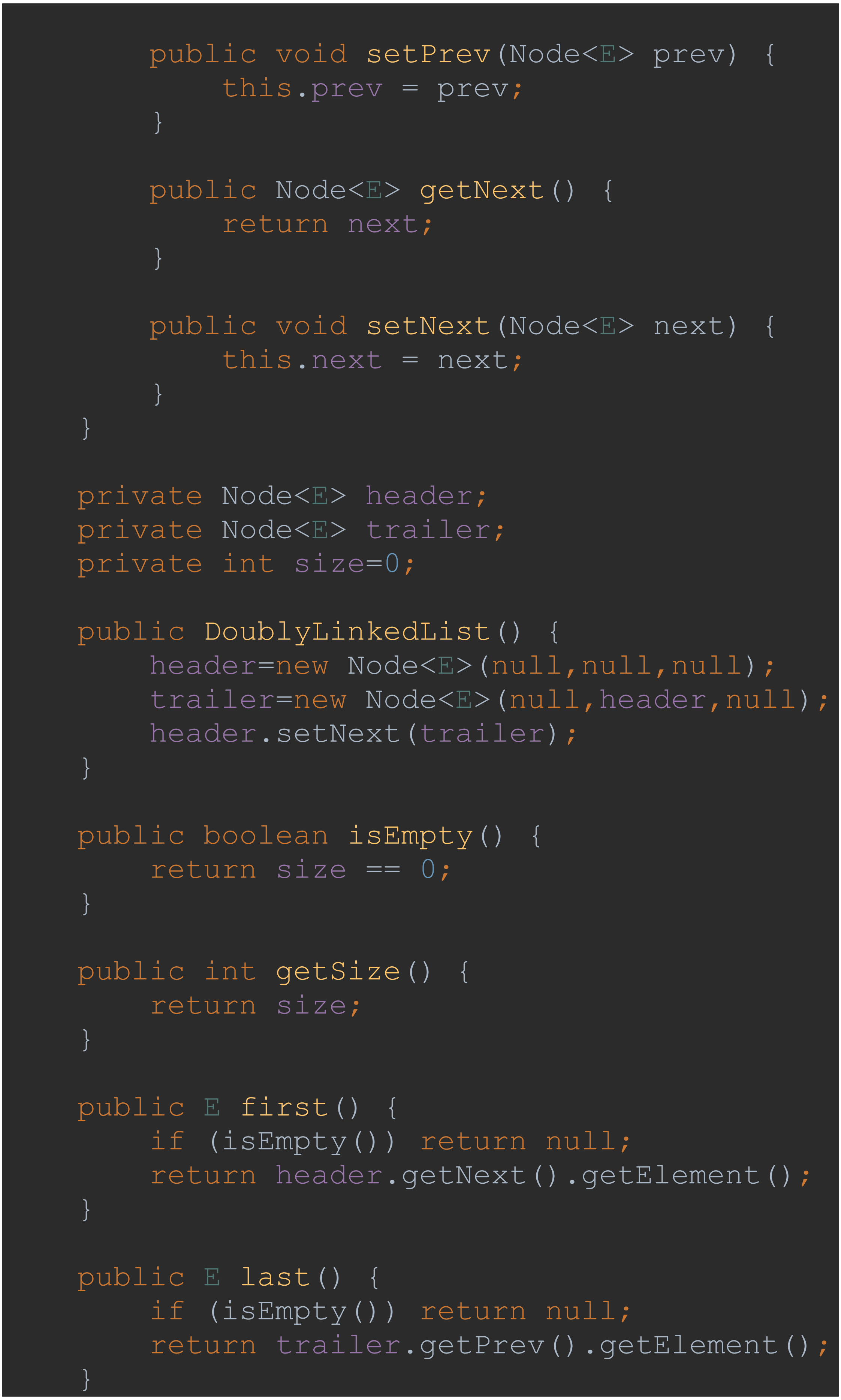
```
    public CircularyLinkedList() {
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public int getSize() {
        return size;
    }

    public E first() {
        if (isEmpty()) return null;
        return tail.getNext().getElement();
    }

    public E last() {
        if (isEmpty()) return null;
        return tail.getElement();
    }

    public void rotate() {
        if (tail != null)
            tail = tail.getNext();
    }

    public void addFirst(E element) {
        if (size == 0) {
            tail = new Node<E>(element, null);
            tail.setNext(tail);//تـؤشر علشان
وتـكون لـنفسها circuly
        } else {
            Node<E> newest = new
Node<E>(element, tail.getNext());
            tail.setNext(newest);

        }
        size++;
    }

    public void addLast(E element) {
```



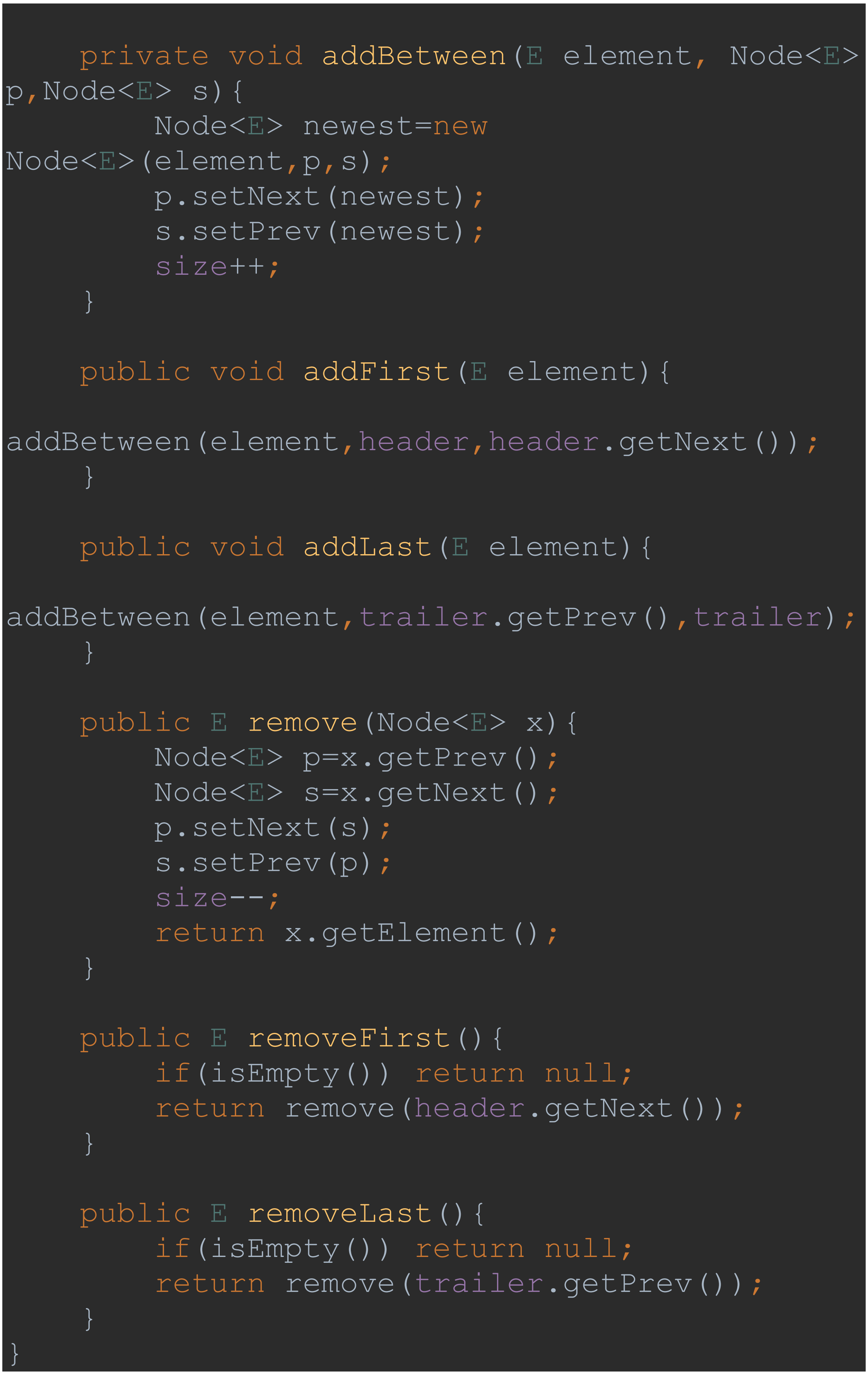
```
        addFirst(element);
        tail = tail.getNext();
    }

    public E removeFirst() {
        if (isEmpty()) {
            return null;
        }
        Node<E> x = tail.getNext();
        if (x == tail) {
            tail = null;
        } else {
            tail.setNext(x.getNext());
        }
        size--;
        return x.getElement();
    }
}
```



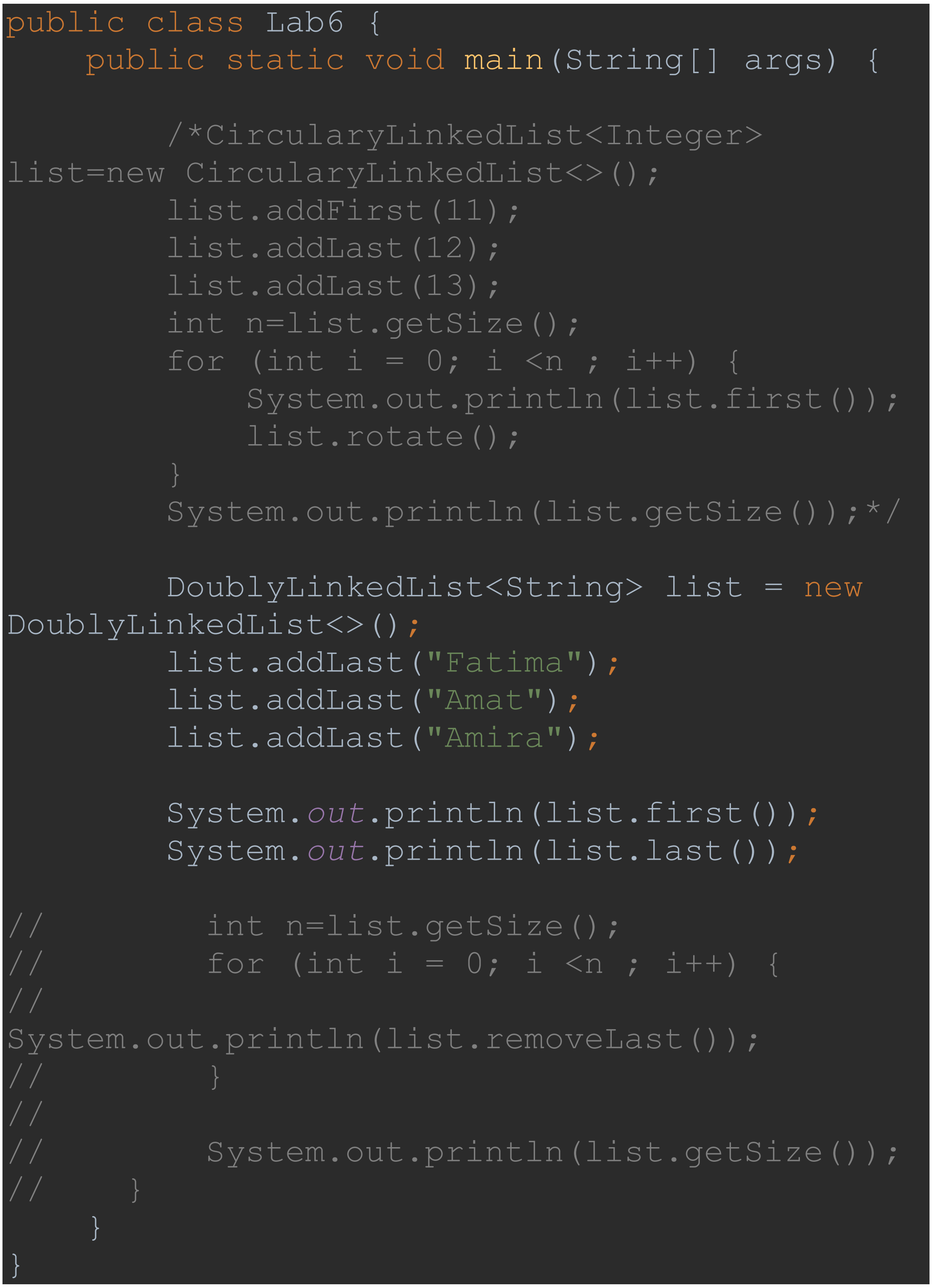
```
public class DoublyLinkedList<E> {

    private static class Node<E>{
        private E element;
        private Node<E> prev;
        private Node<E> next;

        public Node(E element, Node<E> prev,
Node<E> next) {
            this.element = element;
            this.prev = prev;
            this.next = next;
        }

        public E getElement() {
            return element;
        }

        public Node<E> getPrev() {
            return prev;
        }
```



```
        public void setPrev(Node<E> prev) {
            this.prev = prev;
        }

        public Node<E> getNext() {
            return next;
        }

        public void setNext(Node<E> next) {
            this.next = next;
        }
    }

    private Node<E> header;
    private Node<E> trailer;
    private int size=0;

    public DoublyLinkedList() {
        header=new Node<E>(null,null,null);
        trailer=new Node<E>(null,header,null);
        header.setNext(trailer);
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public int getSize() {
        return size;
    }

    public E first() {
        if (isEmpty()) return null;
        return header.getNext().getElement();
    }

    public E last() {
        if (isEmpty()) return null;
        return trailer.getPrev().getElement();
    }
```

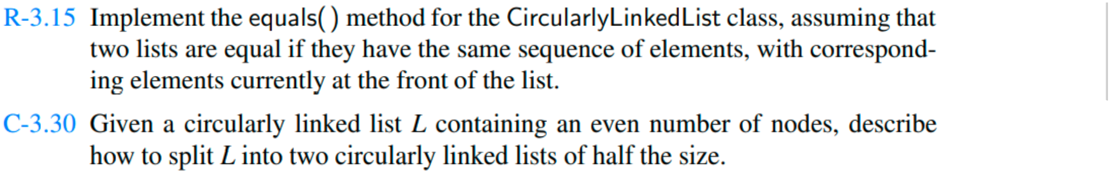


```
    private void addBetween(E element, Node<E>
p,Node<E> s){
        Node<E> newest=new
Node<E>(element,p,s);
        p.setNext(newest);
        s.setPrev(newest);
        size++;
    }

    public void addFirst(E element){

addBetween(element,header,header.getNext());
    }

    public void addLast(E element){

addBetween(element,trailer.getPrev(),trailer);
    }

    public E remove(Node<E> x){
        Node<E> p=x.getPrev();
        Node<E> s=x.getNext();
        p.setNext(s);
        s.setPrev(p);
        size--;
        return x.getElement();
    }

    public E removeFirst(){
        if(isEmpty()) return null;
        return remove(header.getNext());
    }

    public E removeLast(){
        if(isEmpty()) return null;
        return remove(trailer.getPrev());
    }
}
```



```
public class Lab6 {
    public static void main(String[] args) {

        /*CircularyLinkedList<Integer>
list=new CircularyLinkedList<>();
        list.addFirst(11);
        list.addLast(12);
        list.addLast(13);
        int n=list.getSize();
        for (int i = 0; i <n ; i++) {
            System.out.println(list.first());
            list.rotate();
        }
        System.out.println(list.getSize());*/

        DoublyLinkedList<String> list = new
DoublyLinkedList<>();
        list.addLast("Fatima");
        list.addLast("Amat");
        list.addLast("Amira");

        System.out.println(list.first());
        System.out.println(list.last());

//          int n=list.getSize();
//          for (int i = 0; i <n ; i++) {
//
System.out.println(list.removeLast());
//          }
//
//          System.out.println(list.getSize());
//      }
    }
}
```

Task6:

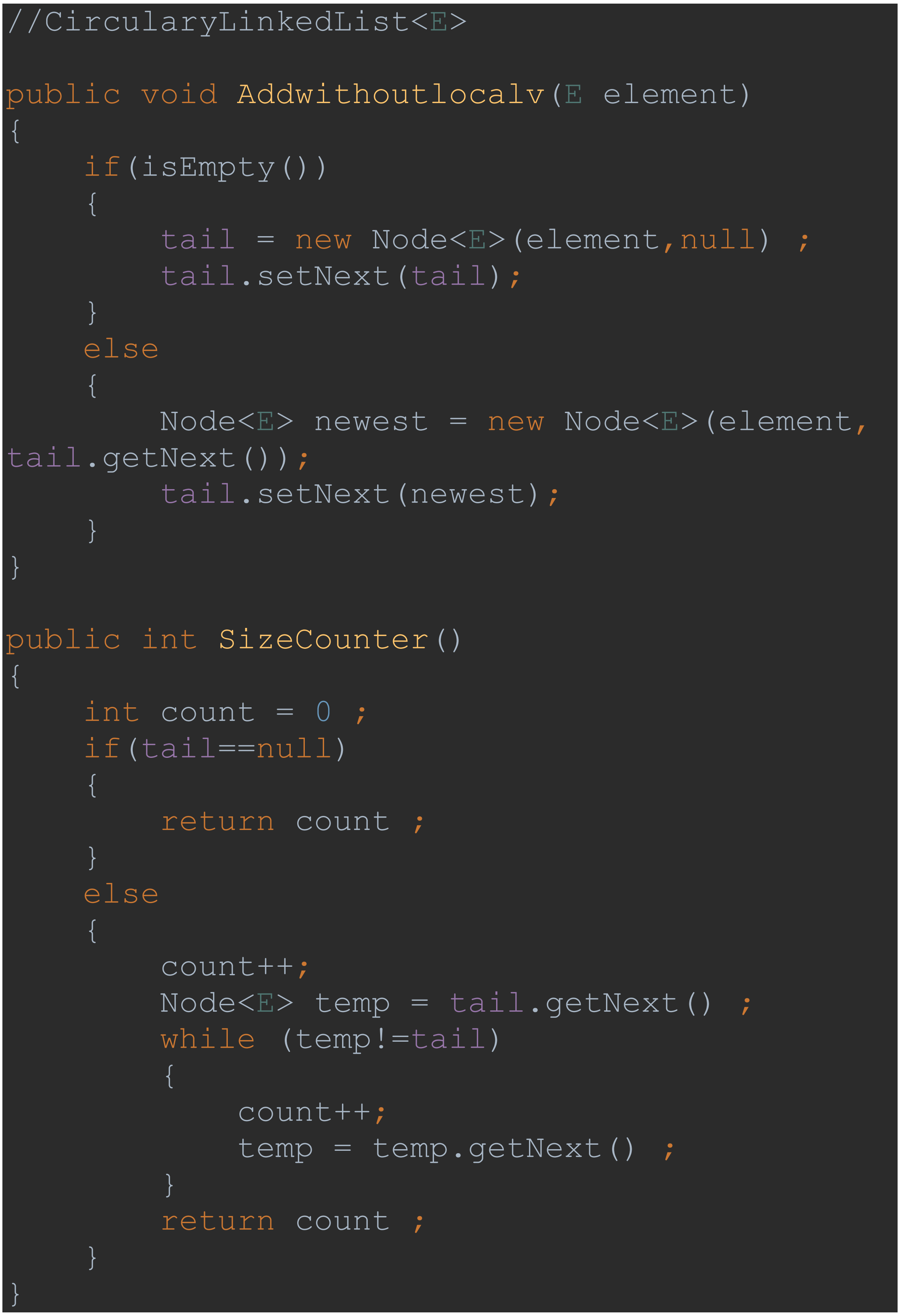


R-3.15 Implement the equals( ) method for the CircularlyLinkedList class, assuming that two lists are equal if they have the same sequence of elements, with corresponding elements currently at the front of the list.

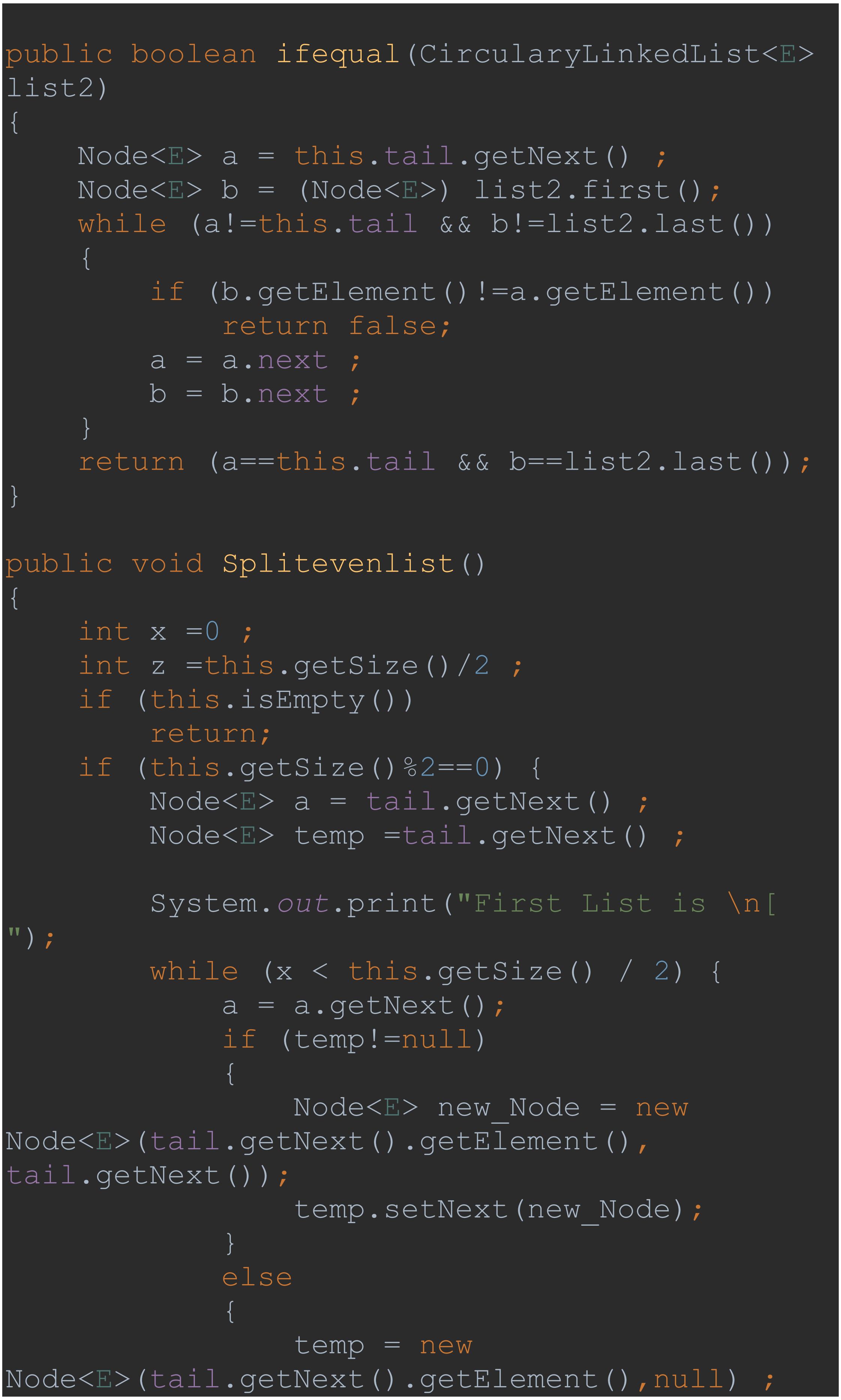
C-3.30 Given a circularly linked list *L* containing an even number of nodes, describe how to split *L* into two circularly linked lists of half the size.

R-3.7 Consider the implementation of CircularlyLinkedList.addFirst, in Code Fragment 3.16. The else body at lines 39 and 40 of that method relies on a locally declared variable, newest. Redesign that clause to avoid use of any local variable.

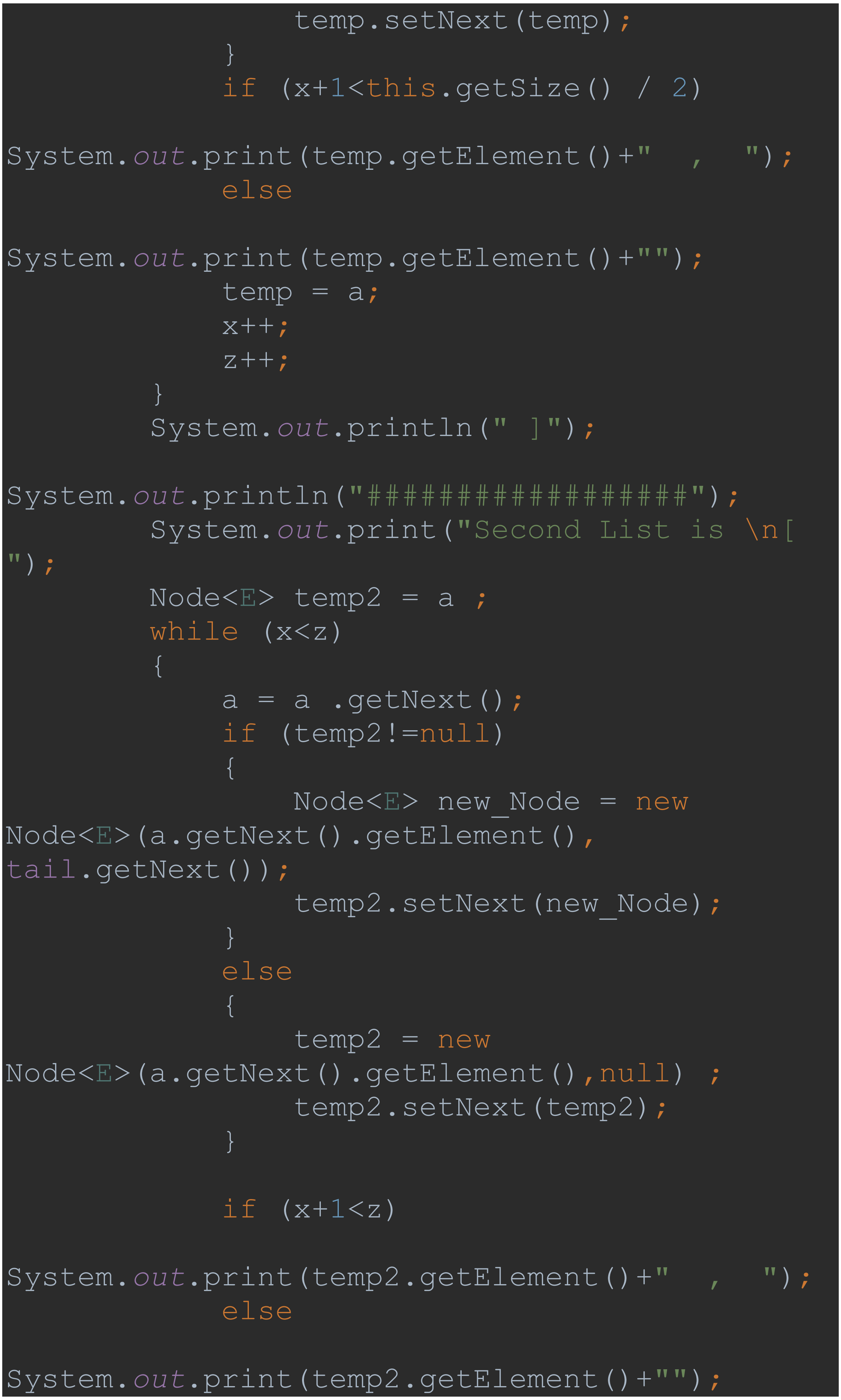
C-3.35 Implement the clone( ) method for the DoublyLinkedList class.



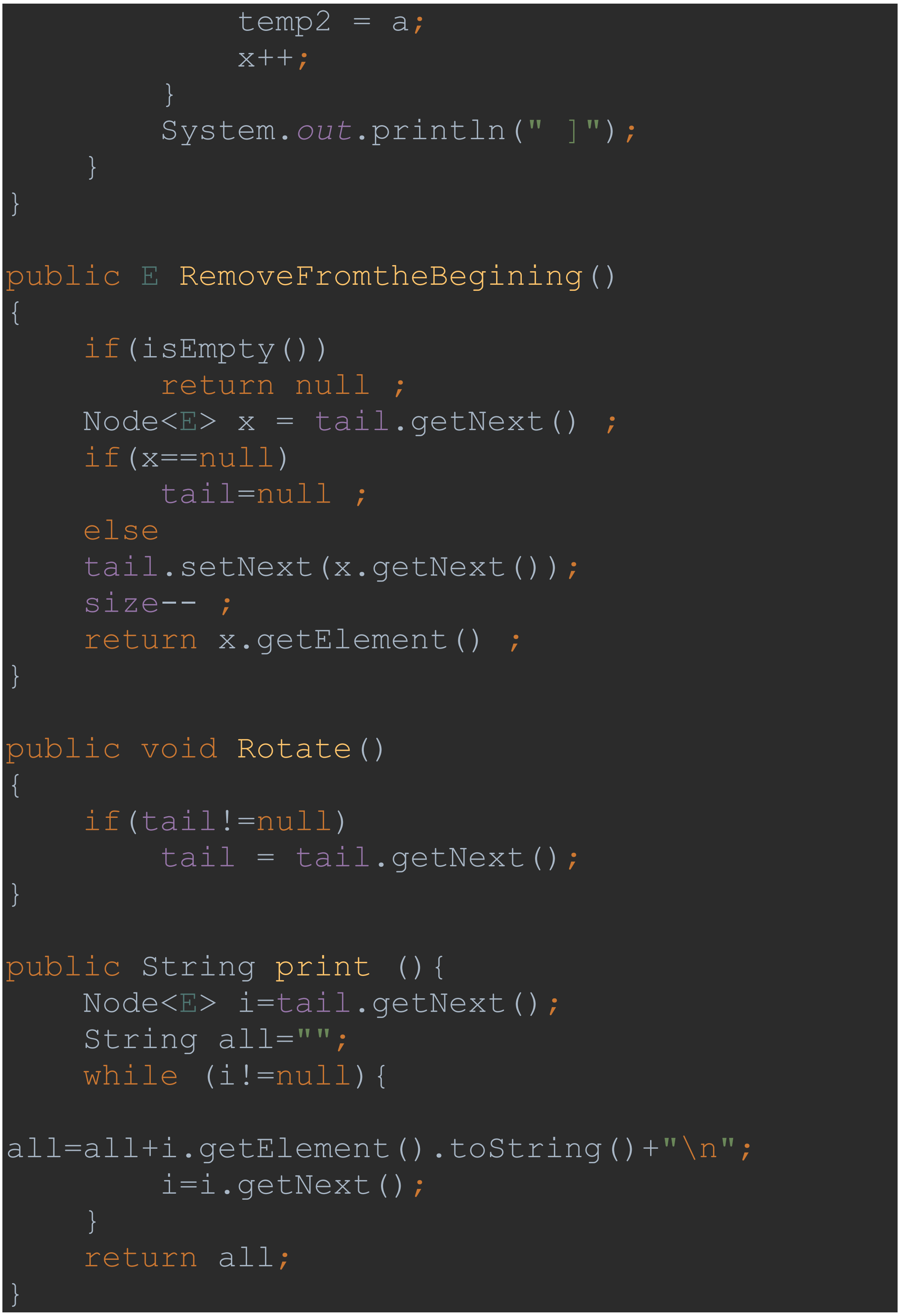
```
//CircularyLinkedList<E>

public void Addwithoutlocalv(E element)
{
     if(isEmpty())
     {
          tail = new Node<E>(element,null) ;
          tail.setNext(tail);
     }
     else
     {
          Node<E> newest = new Node<E>(element,
tail.getNext());
          tail.setNext(newest);
     }
}

public int SizeCounter()
{
     int count = 0 ;
     if(tail==null)
     {
          return count ;
     }
     else
     {
          count++;
          Node<E> temp = tail.getNext() ;
          while (temp!=tail)
          {
               count++;
               temp = temp.getNext() ;
          }
          return count ;
     }
}
```



```
public boolean ifequal(CircularyLinkedList<E>
list2)
{
    Node<E> a = this.tail.getNext() ;
    Node<E> b = (Node<E>) list2.first();
    while (a!=this.tail && b!=list2.last())
    {
        if (b.getElement()!=a.getElement())
            return false;
        a = a.next ;
        b = b.next ;
    }
    return (a==this.tail && b==list2.last());
}

public void Splitevenlist()
{
    int x =0 ;
    int z =this.getSize()/2 ;
    if (this.isEmpty())
        return;
    if (this.getSize()%2==0) {
        Node<E> a = tail.getNext() ;
        Node<E> temp =tail.getNext() ;

        System.out.print("First List is \n[
");
        while (x < this.getSize() / 2) {
            a = a.getNext();
            if (temp!=null)
            {
                Node<E> new_Node = new
Node<E>(tail.getNext().getElement(),
tail.getNext());
                temp.setNext(new_Node);
            }
            else
            {
                temp = new
Node<E>(tail.getNext().getElement(),null) ;
```



```
                temp.setNext(temp);
            }
            if (x+1<this.getSize() / 2)
                
System.out.print(temp.getElement()+"  ,  ");
            else
                
System.out.print(temp.getElement()+"");
            temp = a;
            x++;
            z++;
        }
        System.out.println(" ]");
        
System.out.println("##################");
        System.out.print("Second List is \n[
");
        Node<E> temp2 = a ;
        while (x<z)
        {
            a = a .getNext();
            if (temp2!=null)
            {
                Node<E> new_Node = new 
Node<E>(a.getNext().getElement(), 
tail.getNext());
                temp2.setNext(new_Node);
            }
            else
            {
                temp2 = new 
Node<E>(a.getNext().getElement(),null) ;
                temp2.setNext(temp2);
            }

            if (x+1<z)
                
System.out.print(temp2.getElement()+"  ,  ");
            else
                
System.out.print(temp2.getElement()+"");
```



```
            temp2 = a;
            x++;
        }
        System.out.println(" ]");
    }
}

public E RemoveFromtheBegining()
{
    if(isEmpty())
        return null ;
    Node<E> x = tail.getNext() ;
    if(x==null)
        tail=null ;
    else
    tail.setNext(x.getNext());
    size-- ;
    return x.getElement() ;
}

public void Rotate()
{
    if(tail!=null)
        tail = tail.getNext();
}

public String print (){
    Node<E> i=tail.getNext();
    String all="";
    while (i!=null){

all=all+i.getElement().toString()+"\n";
        i=i.getNext();
    }
    return all;
}
```

Test1:

```
public class Task6 {
        public static void main(String[] args)
{
            CircularyLinkedList<Integer> list
=new CircularyLinkedList<>();
            list.addLast(11);
            list.addLast(12);
            list.addLast(13);
            list.addLast(14);
            CircularyLinkedList<Integer> list2
=new CircularyLinkedList<>();
            list.addLast(1);
            list.addLast(2);
            list.addLast(3);
            list.addLast(4);

            if (list.ifequal(list2)==true)
                System.out.println("The lists
are equal .");
            else
            System.out.println("They are not
equal .");
        }

}
```

Output:

They are not equal.

Test2:

```
public class Adding {
        public static void main(String[] args)
{
            CircularyLinkedList<Integer> list
=new CircularyLinkedList<>();
            list.addLast(11);
            list.addLast(12);
            list.Addwithoutlocalv(13);
            list.Addwithoutlocalv(14);
            list.AddwithoutlocaIv(15);
            System.out.println(list.print());
        }
}
```

Test3:

```
public class SizeCounter {
        public static void main(String[] args)
{
            CircularyLinkedList<Integer> list
=new CircularyLinkedList<>();
            list.addFirst(1);
            list.addFirst(22);
            list.addFirst(233);
            list.addFirst(26);
            list.addFirst(20);
            list.addFirst(100);
            list.addFirst(10);

System.out.println(list.SizeCounter());
        }
}
```

Output:

```
7
```

Test4:

```
public class Spilt {
        public static void main(String[] args)
{
            CircularyLinkedList<Integer> list
= new CircularyLinkedList<>();
            list.addFirst(1);
            list.addFirst(2);
            list.addFirst(3);
            list.addFirst(4);
            list.addFirst(5);
            list.addFirst(6);
            list.Splitevenlist();
        }
}
```

Output:

```
First List is

[ 4 , 5 , 6 ]

####################

Second List is

[ 1 , 2 , 3 ]
```

```
//DoublyLinkedList<E>

public void FindMiddle()
{
    Node<E> temp =  header ;
    int c = 0 ;
    while (temp!=null)
    {
        c++ ;
        temp = temp.getNext() ;
    }
    temp = header ;
    int  p = 1 ;

    int mid = (c+1)/2 ;
    while (temp!=null)
    {
        if (p==mid)
            break;
        p++;
        temp = temp.getNext() ;
    }
    System.out.println("The Middle Element is
: "+temp.getElement());
}
```

```
public int Counter()
{
    int s = 0 ;
    Node<E> temp = header ;
    if (header.getNext()==trailer)
        return s ;
    while (temp!=trailer)
    {
        s++;
        temp = temp.getNext();
    }
    return s ;
}

public boolean ifequal(DoublyLinkedList<E>
list)
{
    Node<E> a = this.header.getNext();
    Node<E> b = list.header.getNext();
    while (a!=this.trailer && b!=list.trailer)
    {
        if (a.getElement()!=b.getElement())
            return false ;
        a = a.getNext();
        b = b.getNext();
    }
    return (a==this.trailer &&
b==list.trailer);
}
```

Test1:

```
public class Counter {
        public static void main(String[] args)
{
            DoublyLinkedList<Integer> list =
new DoublyLinkedList<>();
            list.addLast(1);
            list.addLast(2);
            list.addLast(3);
            list.addLast(4);
            list.addLast(5);
            list.addLast(6);
            list.addLast(7);
            list.addLast(9);
            list.addLast(10);
            System.out.println("The Size of
DoublyLinked List is :"+list.Counter());
        }
}
```

Output:

The Size of DoublyLinked List is :10

Test2:

```
public class Equal {
        public static void main(String[] args)
{
            DoublyLinkedList<Integer> list =
new DoublyLinkedList<>();
            list.addLast(1);
            list.addLast(2);
            list.addLast(3);
            list.addLast(4);
            DoublyLinkedList<Integer> list2 =
new DoublyLinkedList<>();
            list2.addLast(1);
            list2.addLast(2);
            list2.addLast(3);
            list2.addLast(4);
            if (list.ifequal(list2)==true)
                System.out.println("The Doubly
Linked Lists Are Equal . ");
            else
                System.out.println("The Doubly
Linked Lists Aren't Equal . ");
```

```
        }

}
```

Output:

The Doubly Linked Lists Are Equal .

Test3:

```
public class MiddleNode {
        public static void main(String[] args) {
            DoublyLinkedList<Integer> D_list = new DoublyLinkedList<>();
            D_list.addLast(1);
            D_list.addLast(2);
            D_list.addLast(3);
            D_list.addLast(4);
            D_list.addLast(5);
            D_list.addLast(6);
            D_list.addLast(7);
            D_list.addLast(9);
            D_list.addLast(10);
            D_list.FindMiddle();
        }
}
```

Output:

The Middle Element is : 5.