```java
package Wyjatki;

public class EmptyQueueException extends Throwable {
}
```

ZADANIE 1

KLASA ELEMENT

```java
package Zadanie1;

import java.util.Objects;

public class Element<T> {
    private T data;
    private Element<T> nextElem;

    public Element(T data) {
        this.data = data;
    }

    public T getData() {
        return data;
    }

    public void setData(T data) {
        this.data = data;
    }

    public Element<T> getNextElem() {
        return nextElem;
    }

    public void setNextElem(Element<T> nextElem) {
        this.nextElem = nextElem;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Element)) return false;
        Element<?> element = (Element<?>) o;
        return Objects.equals(data, element.data);
    }

    @Override
    public int hashCode() {
        return Objects.hash(data);
    }
}
```

INTERFEJS LinkedList

```java
package Zadanie1;

public interface LinkedListOneWay<E> {
    void addEnd(E e);
    void insert(int pos, E e);
    E get( int pos );
```

```
    E set(int pos, E e);
    boolean contains( E e );
    int size();
    void clear();
    void deletePos( int pos);
    boolean delete( E e );
    E deleteEl(Element e);
    void wyswietlListe();
}
```

IMPLEMENTACJA

```
package Zadanie1;

public class OneWayLinkedListWithoutSentinel<T> implements
LinkedListOneWay<T>{
    Element<T> head = null;

    public OneWayLinkedListWithoutSentinel() {}

    @Override
    public void addEnd(T t) {
        Element<T> newElem = new Element<>(t);
        if(head==null){
            head=newElem;
            newElem.setNextElem(newElem);
        }
        else{
            Element<T> tail = head;
            while(tail.getNextElem() != head){
                tail = tail.getNextElem();
            }
            tail.setNextElem(newElem);
            newElem.setNextElem(head);
        }
    }

    @Override
    public void insert(int pos, T t) {

        Element<T> actElem = head;
        Element<T> newElem = new Element<>(t);

        if(pos == 0){
            newElem.setNextElem(head);

            while (actElem.getNextElem()!=head){
                actElem = actElem.getNextElem();
            }
            head = newElem;
            actElem.setNextElem(newElem);

        }
        else{
            for(int i =0; i<pos-1;i++){
                actElem = actElem.getNextElem();
            }
            newElem.setNextElem(actElem.getNextElem());
            actElem.setNextElem(newElem);
        }
```

```
}

@Override
public T get(int pos) {
    Element<T> actElem = head;
    for (int i = 0; i < pos; i++) {
        actElem = actElem.getNextElem();
    }
    return actElem.getData();
}

@Override
public T set(int pos, T t) {
    Element<T> actElem = head;
    for (int i = 0; i < pos; i++) {
        actElem = actElem.getNextElem();
    }
    T valBefore = actElem.getData();
    actElem.setData(t);
    return valBefore;
}

@Override
public boolean contains(T t) {
    Element<T> actElem = head.getNextElem();
    while (actElem!=head){
        if(actElem.getData()==t) return true;
        actElem = actElem.getNextElem();
    }
    return false;
}

@Override
public int size() {
    int pos=1;
    Element<T> actElem=head;

    if(actElem==null) return 0;

    actElem=actElem.getNextElem();
    while(actElem!=head)
    {
        pos++;
        actElem=actElem.getNextElem();
    }
    return pos;
}

@Override
public void clear() {
    head = null;
}

@Override
public void deletePos(int pos) {
    Element<T> actElem = head;
    if(pos == 0 || pos%size()==0){
        head = head.getNextElem();
        for (int i = 0; i < size()-1; i++) {
```

```
                actElem = actElem.getNextElem();
            }
            actElem.setNextElem(head);
        }else{
            for (int i = 0; i < pos-1; i++) {
                actElem = actElem.getNextElem();
            }
            actElem.setNextElem(actElem.getNextElem().getNextElem());
        }
    }

    @Override
    public boolean delete(T t) {
        Element<T> actElem = head;

        if (actElem.getData() == t) {
            for (int i = 0; i < size() - 1; i++) {
                actElem = actElem.getNextElem();
            }
            actElem.setNextElem(head.getNextElem());
            head = head.getNextElem();
            return true;
        }

        for (int i = 0; i < size() - 1; i++) {
            if (actElem.getNextElem().getData() == t) {
                actElem.setNextElem(actElem.getNextElem().getNextElem());
            }
            actElem = actElem.getNextElem();
        }
        return false;
    }

    @Override
    public T deleteEl(Element e) {
        Element<T> actElem = head.getNextElem();
        T data = null;
        if(head == e){
            data = head.getData();
            head = head.getNextElem();
            for (int i = 0; i < size(); i++) {
                actElem = actElem.getNextElem();
            }
            actElem.setNextElem(head);
        }else{
            actElem = head;
            while(actElem.getNextElem()!=e)
            {
                actElem=actElem.getNextElem();
            }
            data = actElem.getNextElem().getData();

            actElem.setNextElem(actElem.getNextElem().getNextElem());
        }

        return data;
    }

    @Override
    public void wyswietlListe() {
        Element<T> actElem = head;
```

```
        if(size()==0) System.out.println("[]");

        else{
            System.out.print('[');
            for (int i = 0; i < size(); i++) {
                System.out.print(actElem.getData()+", ");
                actElem = actElem.getNextElem();
            }
            System.out.print("]\n");
        }

    }
}
```

TEST

```
package Zadaniel;

public class OneWayTest {
    public static void main(String[] args) {
        OneWayLinkedListWithoutSentinel<Integer> list = new
OneWayLinkedListWithoutSentinel<>();
        list.addEnd(0);
        list.addEnd(1);
        list.addEnd(2);
        list.addEnd(3);
        list.addEnd(4);
        list.addEnd(5);
        list.addEnd(6);
        System.out.println(list.get(3));
        list.wyswietlListe();
        System.out.println();
        System.out.println("AFTER");
        list.deletePos(0);
        list.deletePos(1);
        list.wyswietlListe();
        System.out.println("\nSET NA POZYCJI");
        list.set(3,10);
        list.set(0,10);
        list.wyswietlListe();
        System.out.println("\nINSERT NA POZYCJI");
        list.insert(0,20);
        list.insert(1,21);
        list.insert(5,21);
        list.wyswietlListe();
        System.out.println();
        System.out.println(list.contains(21));
        System.out.println(list.contains(6));
        System.out.println(list.contains(22222));
        System.out.println(list.size());
        list.clear();
        System.out.println(list.size());
        list.addEnd(0);
        list.addEnd(1);
        list.addEnd(2);
        list.addEnd(3);
        list.addEnd(4);
        list.addEnd(5);
        list.addEnd(6);
        list.wyswietlListe();
```

```
        list.delete(0);
        System.out.println();
        list.wyswietlListe();
        list.delete(1);
        System.out.println();
        list.wyswietlListe();

    }
}
```

WYNIKI:

3

[0, 1, 2, 3, 4, 5, 6,]

AFTER

[1, 3, 4, 5, 6,]

SET NA POZYCJI

[10, 3, 4, 10, 6,]

INSERT NA POZYCJI

[20, 21, 10, 3, 4, 21, 10, 6,]

true

true

false

8

0

[0, 1, 2, 3, 4, 5, 6,]

[1, 2, 3, 4, 5, 6,]

[2, 3, 4, 5, 6,]

ZADANIE 2

Klasa Person

```
package Zadanie2;

public class Person {
    private int number;
    private boolean wasMentioned;

    public Person(int number) {
        this.number = number;
        this.wasMentioned = false;
    }
```

```
    public int getNumber() {
        return number;
    }

    public void setNumber(int number) {
        this.number = number;
    }

    public boolean isWasMentioned() {
        return wasMentioned;
    }

    @Override
    public String toString() {
        return ""+number;
    }

    public void setWasMentioned(boolean wasMentioned) {
        this.wasMentioned = wasMentioned;
    }
}
```

PersonCircle

```
package Zadanie2;

import Zadanie1.OneWayLinkedListWithoutSentinel;

public class PersonCircle {
    public static OneWayLinkedListWithoutSentinel<Person> persons = new
OneWayLinkedListWithoutSentinel<>();

    public static void appendPersons(int amount){
        for (int i = 1; i <= amount; i++) {
            persons.addEnd(new Person(i));
        }
    }

    public static void deletePersonWithStep(int step){
        int personsIter = 0;
        int soutCount = 0;
        for (int i = 0; soutCount<persons.size()-1; i++) {

            if(!persons.get(i).isWasMentioned()){
                personsIter++;

                if((personsIter)%step==0){
                    persons.get(i).setWasMentioned(true);
                    System.out.print(persons.get(i).getNumber()+",");
                    soutCount++;
                }
            }


        }
    }

    public static void circle(int n, int k){
        appendPersons(n);
        deletePersonWithStep(k);
```

```
    }

    public static void main(String[] args) {
        circle(10,2);
    }

}
```

KONSOLA:

2,4,6,8,10,3,7,1,9,

ZADANIE 3

Interfejs:

```
package Zadanie3;

import Wyjatki.EmptyQueueException;

public interface IQueue<T> {
    boolean isEmpty();
    boolean isFull();
    T dequeue() throws EmptyQueueException;
    void enqueue(T elem);
    T first() throws EmptyQueueException;
}
```

Queue

```
package Zadanie3;

import Wyjatki.EmptyQueueException;
import Zadanie1.OneWayLinkedListWithoutSentinel;

import java.util.StringJoiner;

public class Queue<T> implements IQueue<T> {

    OneWayLinkedListWithoutSentinel<T> list = new
OneWayLinkedListWithoutSentinel<>();

    @Override
    public boolean isEmpty() {
        return list.size()==0;
    }

    @Override
    public boolean isFull() {
        return false;
    }

    @Override
    public T dequeue() throws EmptyQueueException {
        T elem = list.get(list.size()-1);
        list.deletePos(list.size()-1);
        return elem;
    }

    @Override
    public void enqueue(T elem) {
```

```
        if(isEmpty()) {list.addEnd(elem);}
        else {list.insert(0,elem);}
    }

    public void display(){
        list.wyswietlListe();
    }

    @Override
    public T first() throws EmptyQueueException {
        if(isEmpty()) throw new EmptyQueueException();
        return list.get(list.size()-1);
    }
}
```

QueueTest

```
package Zadanie3;

import Wyjatki.EmptyQueueException;

public class QueueTest {
    public static void main(String[] args) throws EmptyQueueException {
        Queue<Integer> kolejka = new Queue<>();
        kolejka.enqueue(1);
        kolejka.enqueue(2);
        kolejka.enqueue(3);
        kolejka.display();
        kolejka.dequeue();
        kolejka.display();
        System.out.println(kolejka.first());
    }
}
```

Konsola:

[3, 2, 1,]

[3, 2,]

2

Stack:

```
package Zadanie3;

import Zadanie1.OneWayLinkedListWithoutSentinel;

import java.util.Stack;

public class StackList<T> extends Stack<T> {

    OneWayLinkedListWithoutSentinel<T> list = new
OneWayLinkedListWithoutSentinel<>();

    @Override
    public T push(T item) {
        list.addEnd(item);
        return item;
```

```
    }

    @Override
    public T pop() {
        T elem = list.get(list.size()-1);
        list.deletePos(list.size()-1);
        return elem;
    }

    @Override
    public boolean empty() {
        return list.size()==0;
    }

    public void display(){
        list.wyswietlListe();
    }

}
```

StackTest

```
package Zadanie3;

public class StackTest {
    public static void main(String[] args) {
        StackList<Integer> stack = new StackList<>();
        for (int i = 0; i < 5; i++) {
            stack.push(i);
            stack.display();
        }

        for (int i = 0; i < 3; i++) {
            stack.pop();
            stack.display();
        }

    }
}
```

Konsola:

[0,]

[0, 1,]

[0, 1, 2,]

[0, 1, 2, 3,]

[0, 1, 2, 3, 4,]

[0, 1, 2, 3,]

[0, 1, 2,]

[0, 1,]

ZADANIE 4

Lista z głową:

```
package Zadanie4;

import java.util.AbstractList;

public class TwoWayLinkedListWithHead<T> extends AbstractList<T> {

    private Node<T> head = new Node<>(null);

    public TwoWayLinkedListWithHead() {
    }

    public Node getElement(int index){
        Node actElem=head.getNext();
        while(index>0 && actElem!=null){
            index--;
            actElem=actElem.getNext();
        }
        return actElem;
    }

    public void connectWith(Node elem){
        Node last = getElement(size()-1);
        last.setNext(elem);
        elem.setPrev(last);
    }

    public void connectWithIn(Node left, Node right, int idx){
        Node rightConnection = getElement(idx);
        Node leftConnection = getElement(idx-1);

        leftConnection.setNext(left);
        left.setPrev(leftConnection);

        rightConnection.setPrev(right);
        right.setNext(rightConnection);
    }

    public boolean isEmpty(){
        return head==null;
    }

    public void clear(){
        head = new Node<>(null);
    }

    @Override
    public int size() {
        Node elem = head.getNext();
        if(elem == null) return 0;
        int size = 1;
        while(elem.getNext()!=null){
            size++;
            elem = elem.getNext();
        }

        return size;
    }
```

```java
@Override
public boolean add(T t) {
    Node<T> newElem=new Node<>(t);

    if(head.getNext()==null){
        head.setNext(newElem);
        newElem.setNext(null);
        return true;
    }

    Node tail=head;
    while(tail.getNext()!=null)
        tail=tail.getNext();
    tail.setNext(newElem);
    newElem.setPrev(tail);
    newElem.setNext(null);
    return true;
}

public void add(int index, T data) {
    Node<T> newElem= new Node<>(data);

    if(index==0) {
        newElem.setNext(head.getNext());
        newElem.setPrev(head);
        head.setNext(newElem);
        newElem.getNext().setPrev(newElem);
    }else{
        Node actElem=getElement(index-1);
        newElem.setNext(actElem.getNext());
        newElem.setPrev(actElem);
        actElem.getNext().setPrev(newElem);
        actElem.setNext(newElem);
    }
}


@Override
public int indexOf(Object data) {
    int pos=0;
    Node actElem=head;
    while(actElem!=null)
    {
        if(actElem.getValue().equals(data))
            return pos;
        pos++;
        actElem=actElem.getNext();
    }
    return -1;}

@Override
public boolean contains(Object data) {
    return indexOf(data)>=0;}

@Override
public T get(int index) {
    Node actElem=getElement(index);
    return actElem==null?null: (T) actElem.getValue();
}
```

```
    public T set(int index, T data) {
        Node actElem=getElement(index);
        if(actElem==null)
            return null;
        T elemData= (T) actElem.getValue();
        actElem.setValue(data);
        return elemData;
    }

    @Override
    public T remove(int index) {
        if(head==null)
            return null;
        if(index==0){
            T retValue=head.getValue();
            head=head.getNext();
            return retValue;
        }
        Node actElem=getElement(index-1);
        if(actElem==null || actElem.getNext()==null)
            return null;
        T retValue= (T) actElem.getNext().getValue();
        actElem.setNext(actElem.getNext().getNext());
        return retValue;
    }

    @Override
    public boolean remove(Object value) {
        if(head==null)
            return false;
        if(head.getValue().equals(value)){
            head=head.getNext();
            return true;}
        Node actElem=head;
        while(actElem.getNext()!=null &&
!actElem.getNext().getValue().equals(value))
            actElem=actElem.getNext();
        if(actElem.getNext()==null)
            return false;
        actElem.setNext(actElem.getNext().getNext());
        return true;
    }


    @Override
    public String toString() {
        Node el = head.getNext();
        String txt = "[";
        while(el!=null){
            txt += el.getValue()+", ";
            el = el.getNext();
        }
        txt += "]";
        return txt;
    }
}
```

Lista z strażnikiem

```java
package Zadanie4;

import java.util.AbstractList;
public class TwoWayLinkedListWithSentinel<E> extends AbstractList<E> {
    Node<E> sentinel = null;

    public TwoWayLinkedListWithSentinel() {
        sentinel = new Node<>(null);
//        sentinel.setPrev(sentinel);
//        sentinel.setNext(sentinel);
    }

    public Node getElement(int index){
        Node elem = sentinel.getNext();
        int counter = 0;
        while(elem!=sentinel && counter<index){
            counter++;
            elem=elem.getNext();}
        if(elem==sentinel)
            throw new IndexOutOfBoundsException();
        return elem;
    }

    public Node getElement(E value){
        Node elem=sentinel.getNext();
        int counter=0;
        while(elem!=sentinel && !value.equals(elem.getValue())){
            counter++;
            elem=elem.getNext();
        }
        if(elem==sentinel)
            return null;
        return elem;
    }

    public boolean isEmpty(){
        return sentinel.getNext()==sentinel;
    }

    public void clear(){
        sentinel.setNext(sentinel);
        sentinel.setPrev(sentinel);
    }

    public boolean contains(Object value){
        return indexOf(value) != -1;
    }

    @Override
    public E get(int index) {
        Node elem = getElement(index);
        return (E) elem.getValue();
    }

    public E set(int index, E value) {
        Node elem = getElement(index);
        E retValue= (E) elem.getValue();
        elem.setValue(value);
        return retValue;
    }
```

```
    public boolean add(E value) {
        Node newElem=new Node(value);

        Node elem = sentinel;
        if(sentinel.getNext()==null){
            sentinel.setNext(newElem);
            newElem.setPrev(sentinel);
            return true;
        }

        while(elem.getNext()!=null){
            elem = elem.getNext();
        }

        elem.setNext(newElem);
        newElem.setPrev(elem);
        return true;
    }


//    public void add(int index, E value) {
//        Node<E> newElem=new Node<>(value);
//        if(index==0) {sentinel.setValue(value);}
//        else{
//            Node elem=getElement(index-1);
//            elem.insertAfter(newElem);
//        }
//    }

    public int indexOf(Object value) {
        Node elem=sentinel.getNext();
        int counter=0;
        while(elem!=sentinel && !elem.getValue().equals(value)){
            counter++;
            elem=elem.getNext();}
        if(elem==sentinel)
            return -1;
        return counter;
    }

    public E remove(int index) {
        Node elem=getElement(index);
        elem.remove();
        return (E) elem.getValue();
    }

    public boolean remove(Object value) {
        Node elem= getElement((E) value);
        if(elem==null) return false;
        elem.remove();
        return true;
    }

    public int size() {
        Node elem=sentinel.getNext();
        int counter=0;
        while(elem.getNext()!=null){
            counter++;
            elem=elem.getNext();}
        return counter;
    }
```

```
    @Override
    public String toString() {
        Node elem = sentinel.getNext();
        String txt = "[";
        while (elem!=null){
            txt+=elem.getValue()+", ";
            elem = elem.getNext();
        }
        txt+="]";
        return txt;
    }
}
```

Klasa NODE

```
package Zadanie4;

public class Node<T> {
    private T value;
    private Node next;
    private Node prev;

    public T getValue() { return value; }
    public void setValue(T value) { this.value = value; }
    public Node getNext() {return next;}
    public void setNext(Node next) {this.next = next;}
    public Node getPrev() {return prev;}
    public void setPrev(Node prev) {this.prev = prev;}

    Node(T data){this.value=data;}

    public void insertAfter(Node elem){
        elem.setNext(this.getNext());
        elem.setPrev(this);
        this.getNext().setPrev(elem);
        this.setNext(elem);
    }


    public void remove(){
        this.getNext().setPrev(this.getPrev());
        this.getPrev().setNext(this.getNext());
    }
}
```

TESTY

```
package Zadanie4;

public class Test4 {

    public static TwoWayLinkedListWithSentinel<Integer> sentinel = new
TwoWayLinkedListWithSentinel<>();
    public static TwoWayLinkedListWithHead<Integer> head = new
TwoWayLinkedListWithHead<>();
```

```
    //sentinel list after head list
    public static<T> void connect1(){
        Node firstOfSentinel = sentinel.getElement(0);
        head.connectWith(firstOfSentinel);
        System.out.println(head);
    }

    //sentinel list before elem of head list/ idx as parameter
    public static<T> void connect2(int idx){
        Node firstOfSentinel = sentinel.getElement(0);
        Node lastOfSentinel = sentinel.getElement(sentinel.size());
        head.connectWithIn(firstOfSentinel, lastOfSentinel, idx);
        System.out.println(head);
    }

    public static void main(String[] args) {
        generateData();
        System.out.println("S "+sentinel); //nieparzyste
        System.out.println("H "+head); //parzyste
        connect1();
        sentinel = new TwoWayLinkedListWithSentinel<>();
        head = new TwoWayLinkedListWithHead<>();
        generateData();
        connect2(3);
    }

    private static void generateData() {
        for (int i = 0; i < 20; i++) {
            if(i%2==0){
                head.add(i);
            }else{
                sentinel.add(i);
            }
        }
    }
}
```

Konsola:

S [1, 3, 5, 7, 9, 11, 13, 15, 17, 19,]

H [0, 2, 4, 6, 8, 10, 12, 14, 16, 18,]

[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19,]

[0, 2, 4, 1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 6, 8, 10, 12, 14, 16, 18,]