**ЛАБОРАТОРНА РОБОТА № 8**

з дисципліни

«Розробка програмного забезпечення на Java»

Шейко Р. О.

Групи КІ-21-2

**Завдання на лабораторну роботу**

1. Проаналізувати та виконати всі приклади в теоретичній частині;

2. Відповісти на всі контрольні питання;

3. Напишіть метод класу Queue програми queue.java для виведення вмісту черги. Зауважте, що завдання не зводиться до простого висновку вмісту базового масиву. Вміст черги повинен виводитись від першого вставленого елемента до останнього, а користувач не повинен бачити, що послідовність переривається на межі масиву. Будьте уважні і простежте, щоб один елемент і вміст порожньої черги виводилися коректно незалежно від положення front і rear.

4. Створіть клас Deque за описом деків (двосторонніх черг). Клас повинен містити методи insertLeft(), insertRight(), removeLeft(), removeRight(), isEmpty() та isFull(). Також у ньому має бути реалізована підтримка циклічного перенесення індексів, за аналогією з чергами.

5. Напишіть реалізацію стека з урахуванням класу Deque з п. 4.2. Клас стека повинен підтримувати ті самі методи та можливості, що й клас StackX у програмі stack.java.

1. Увесь код знаходиться у додатках з виконаними завданнями.
2. **Відповіді на контрольні запитання**

1. LIFO та FIFO - це скорочення для принципів управління даними в структурах даних:

- LIFO (Last-In, First-Out) означає, що останній елемент, який був доданий до структури даних, буде першим вилучений. Цей принцип подібний до роботи стека.

- FIFO (First-In, First-Out) означає, що перший елемент, який був доданий до структури даних, буде першим вилучений. Цей принцип подібний до роботи черги.

2. Стек - це структура даних, де дані зберігаються та доступні для операцій відповідно до принципу LIFO. Тобто останній елемент, який був доданий до стеку, буде першим, хто може бути вилучений. Основна операція - це "push" для додавання елементу та "pop" для вилучення останнього доданого елементу.

3. Черга - це структура даних, де дані зберігаються та доступні для операцій відповідно до принципу FIFO. Тобто перший елемент, який був доданий до черги, буде першим, хто може бути вилучений. Основні операції - "enqueue" для додавання елементу та "dequeue" для вилучення першого елементу.

4. Пріоритетна черга - це структура даних, де кожен елемент має призначений пріоритет, і операції вилучення виконуються на основі цього пріоритету. Елементи з більшим пріоритетом вилучаються першими. Принцип роботи полягає в збереженні елементів у відсортованому порядку за пріоритетом.

5. Циклічне перенесення (Circular Shift) використовується в масивах або списках для зсуву елементів вліво або вправо на певну кількість позицій. Це може бути корисним при обробці даних або роботі з кільцевими буферами, де дані повторюються в циклічному порядку.

6. Основні операції у цих структурах даних наступні:

- Стек: push (додавання елементу), pop (вилучення останнього елементу), peek (перегляд останнього елементу).

- Черга: enqueue (додавання елементу), dequeue (вилучення першого елементу), peek (перегляд першого елементу).

- Пріоритетна черга: insert (додавання елементу з певним пріоритетом), remove (вилучення елементу з найвищим пріоритетом).

- Циклічне перенесення: зсув вліво або вправо на певну кількість позицій в масиві або списку.

1. Створимо нову функцію показу:  
   public void displayQueue() {

if (isEmpty()) {

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

return;

}

int current = front;

int count = 0;

while (count < nItems) {

System.out.print(queArray[current] + " ");

current = (current + 1) % maxSize;

count++;

}

System.out.println();

}

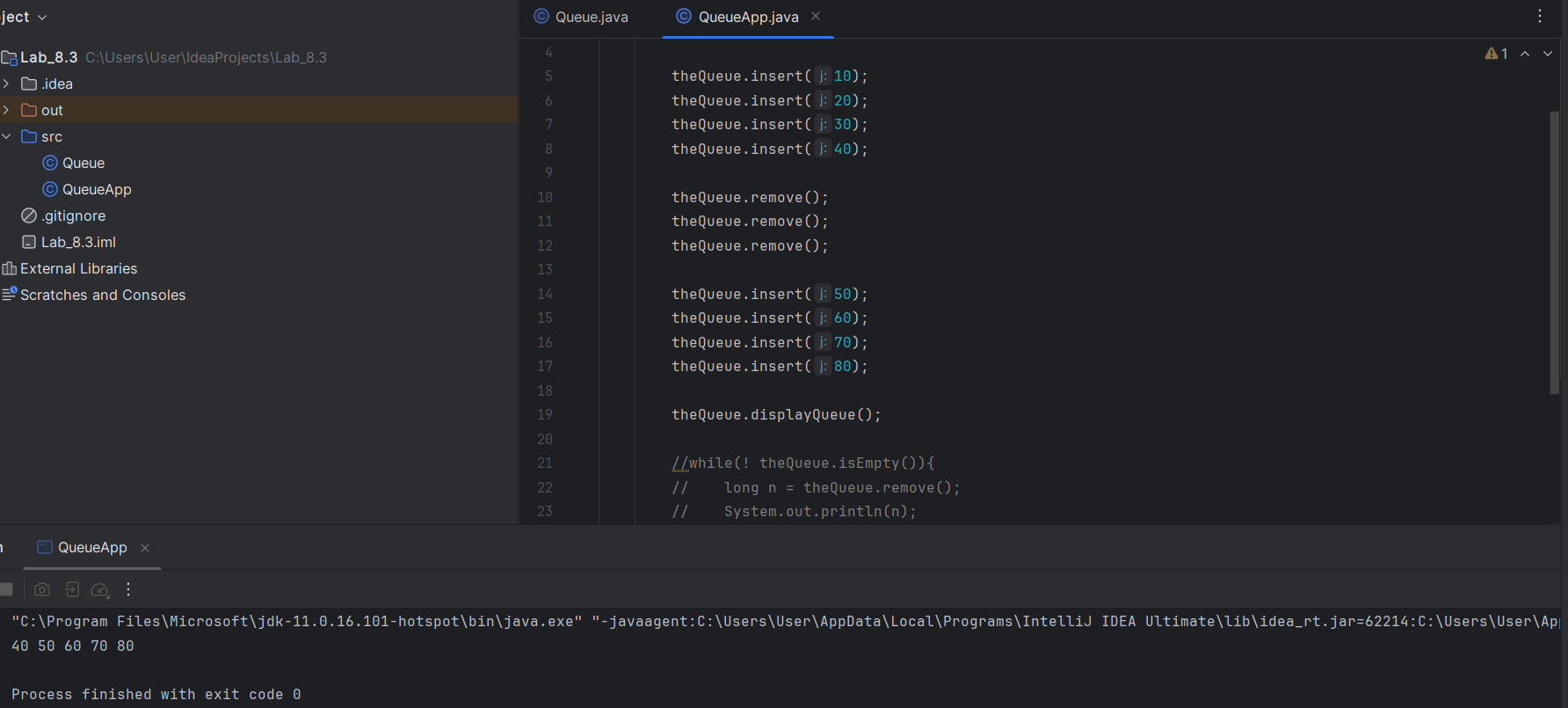


Рис 1. Ілюстрація роботи

1. Створіть клас Deque за описом деків (двосторонніх черг):

У додатку Ї знаходиться код до класу. Ілюстрація роботи на рис 2.:

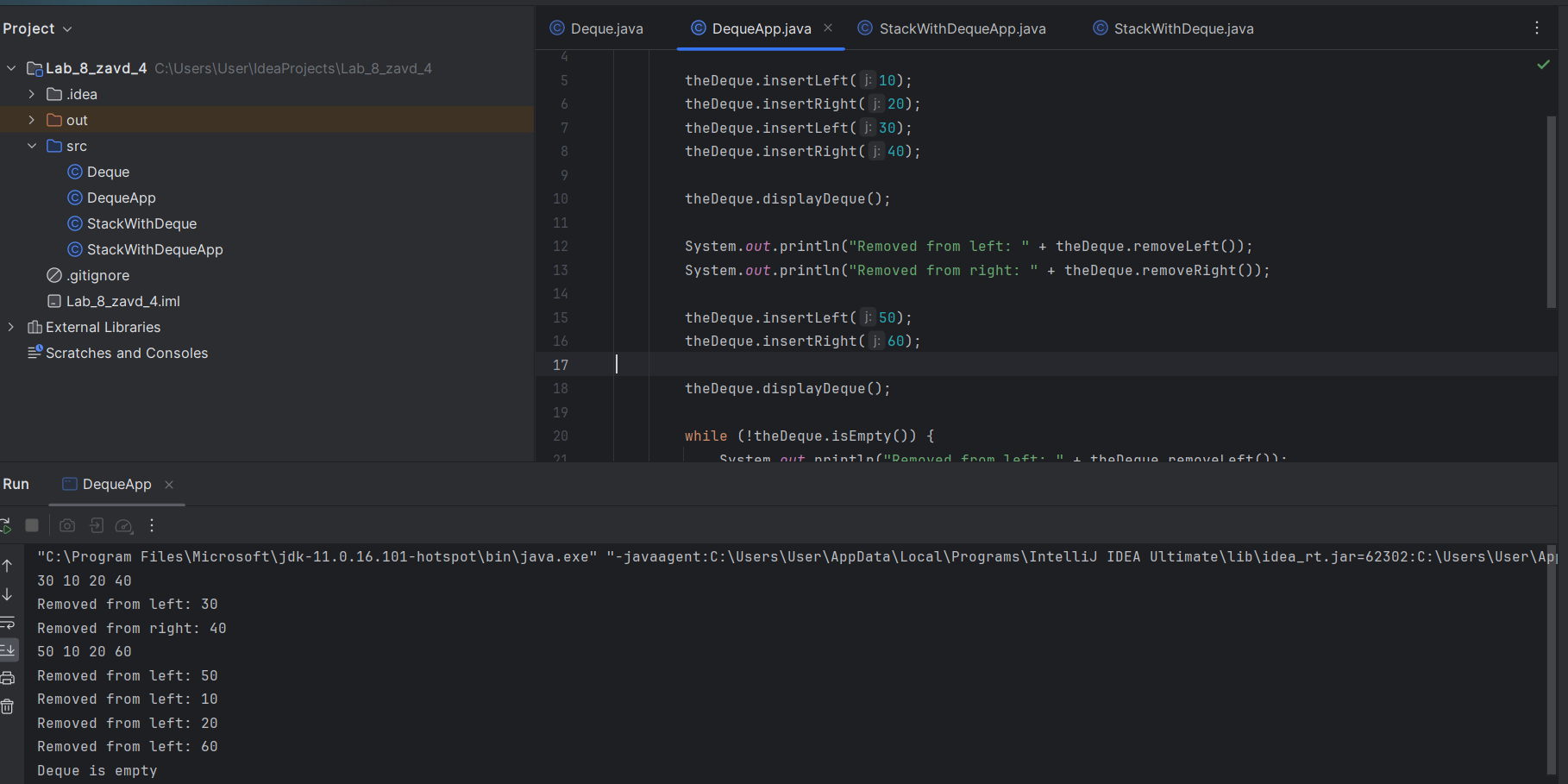


Рис 2. Ілюстрація роботи

1. Напишіть реалізацію стека з урахуванням класу Deque з п. 4.2.

У додатку З, И знаходиться код до класу й програми. Ілюстрація роботи на рис 3:

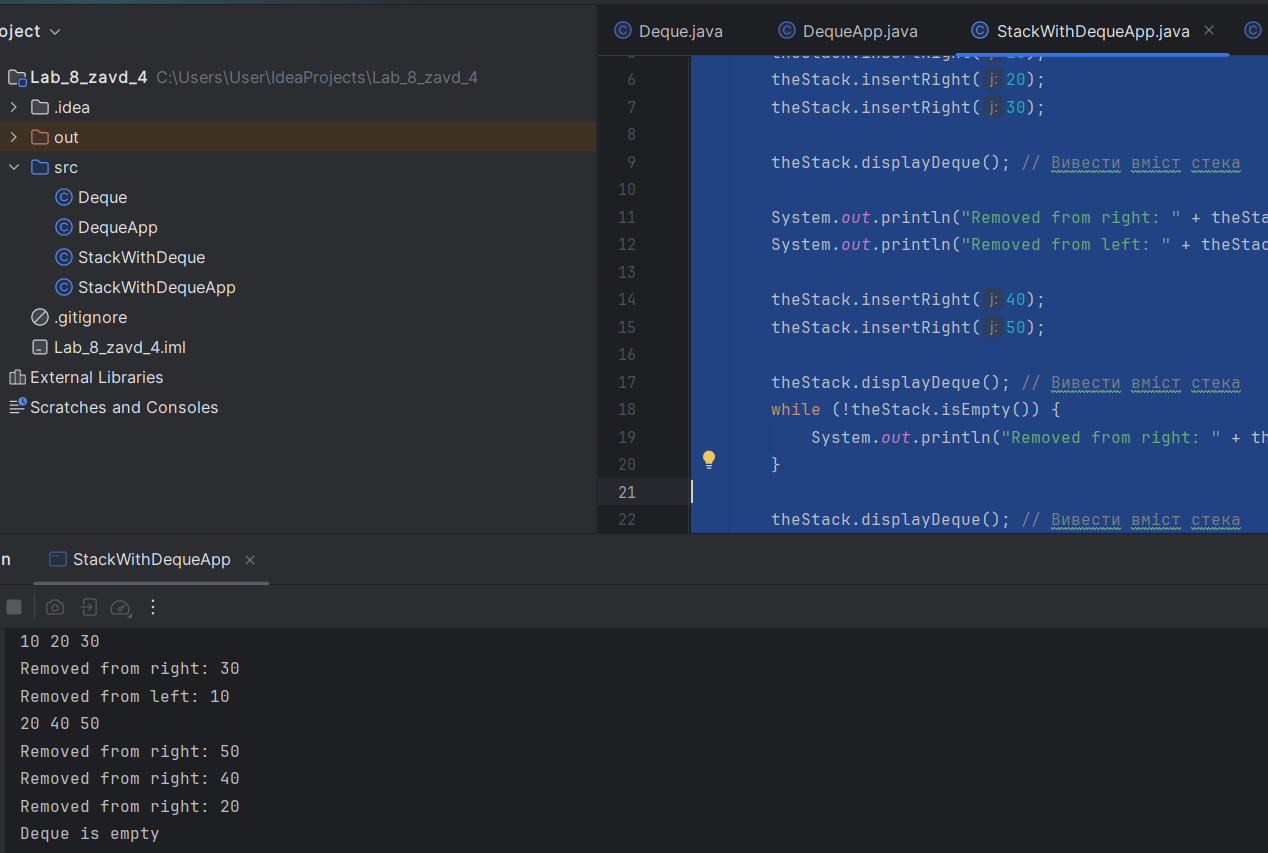


Рис 3. Ілюстрація роботи

**Висновок**

У цій лабораторній роботі було розглянуто основні структури даних: стек, чергу та дек (двосторонню чергу), а також їх взаємозв'язок та практичне використання. Ця лабораторна робота дозволила нам зрозуміти принципи роботи цих структур та навчитися використовувати їх у програмуванні. Відповідаючи на контрольні питання, ми переконалися в нашому розумінні концепцій LIFO та FIFO, стеку, черги, пріоритетної черги та циклічного перенесення.

Для виконання лабораторної роботи були створені наступні компоненти:

1. Метод виведення вмісту черги від першого вставленого елемента до останнього з урахуванням обробки переривань на межі масиву. Важливо бути уважними при реалізації цього методу, щоб забезпечити коректне виведення вмісту черги незалежно від положення front та rear.

2. Створення класу Deque (двосторонньої черги) з методами insertLeft(), insertRight(), removeLeft(), removeRight(), isEmpty() та isFull(). Цей клас також підтримує циклічне перенесення індексів, що дозволяє ефективно використовувати пам'ять для двосторонньої черги.

3. Реалізація стека з використанням класу Deque, де стек підтримує ті самі методи та можливості, що і клас StackX. Це дозволяє легко використовувати двосторонню чергу для створення стеку та використовувати його для управління даними згідно з принципом LIFO.

Завдяки цій лабораторній роботі ми отримали поглиблене розуміння структур даних та навчилися їх реалізовувати в програмах, що важливо для розвитку навичок програмування та роботи зі структурами даних.

Додаток А

*Лістинг файлу «StackX.java»*

public class StackX {

private int maxSize;

private long[] stackArray;

private int top;

public StackX(int s) {

maxSize = s;

stackArray = new long[maxSize];

top = -1;

}

public void push(long j) {

if (isFull())

stackArray[++top] = j;

else

System.out.println("Cant insert, stack is full");

}

public long pop() {

return stackArray[top--];

}

public long peek() {

return stackArray[top];

}

public boolean isEmply() {

return (top == -1);

}

public boolean isFull() {

return !(top == maxSize - 1);

}

}

*Додаток Б*

*Лістинг файлу «Main.java»*

public class Main {

public static void main(String[] args) {

StackX theStack = new StackX(1000);

theStack.push(20);

theStack.push(40);

theStack.push(60);

theStack.push(80);

while (!theStack.isEmply()) {

long value = theStack.pop();

System.out.println(value);

}

}

}

Додаток В

*Лістинг файлу «StackX.java»*

class StackX {

private char[] stackArray;

private int top;

private int maxSize;

public StackX(int max) {

maxSize = max;

stackArray = new char[maxSize];

top = -1;

}

public void push(char j) {

stackArray[++top] = j;

}

public char pop() {

return stackArray[top--];

}

public char peek() {

return stackArray[top];

}

public boolean isEmpty() {

return (top == -1);

}

}

Додаток Г

*Лістинг файлу «Reverser.java»*

class Reverser {

private String input;

private String output;

public Reverser(String in) {

input = in;

}

public String doRev() {

int stackSize = input.length();

StackX theStack = new StackX(stackSize);

for (int j = 0; j < input.length(); j++) {

char ch = input.charAt(j); //Читання символу із вхідного потоку

theStack.push(ch);

}

output = "";

while (!theStack.isEmpty()) {

char ch = theStack.pop();

output = output + ch;

}

return output;

}

}

Додаток [Ґ](https://uk.wikipedia.org/wiki/%D2%90)

*Лістинг файлу «ReverseApp.java»*

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class ReverseApp {

public static void main(String[] args) throws IOException {

String input, output;

while (true) {

System.out.println("Введите строку: ");

System.out.flush();

input = getString();

if (input.equals("")) { break; }

Reverser theReverser = new Reverser(input);

output = theReverser.doRev();

System.out.println("Reverser " + output);

}

}

public static String getString() throws IOException {

InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader br = new BufferedReader(isr);

String s = br.readLine();

return s;

}

}

Додаток Д

*Лістинг файлу «*QueueApp*.java»*

public class QueueApp {

public static void main(String[] args) {

Queue theQueue = new Queue(5);

theQueue.insert(10);

theQueue.insert(20);

theQueue.insert(30);

theQueue.insert(40);

theQueue.remove();

theQueue.remove();

theQueue.remove();

theQueue.insert(50);

theQueue.insert(60);

theQueue.insert(70);

theQueue.insert(80);

theQueue.displayQueue();

//while(! theQueue.isEmpty()){

// long n = theQueue.remove();

// System.out.println(n);

// System.out.println(" ");

//}

}

}

Додаток Е

*Лістинг файлу «Queue.java»*

class Queue {

private int maxSize;

private long[] queArray;

private int front;

private int rear;

private int nItems;

public Queue(int s) {

maxSize = s;

queArray = new long[maxSize];

front = 0;

rear = -1;

nItems = 0;

}

public void insert(long j) {

if (rear == maxSize - 1) {

rear = -1;

}

queArray[++rear] = j;

nItems++;

}

public long remove() {

long temp = queArray[front++];

if (front == maxSize) {

front = 0;

}

nItems--;

return temp;

}

public long peekFront() {

return queArray[front];

}

public boolean isEmpty() {

return (nItems == 0);

}

public boolean isFull() {

return (nItems == maxSize);

}

public int size() {

return nItems;

}

public void displayQueue() {

if (isEmpty()) {

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

return;

}

int current = front;

int count = 0;

while (count < nItems) {

System.out.print(queArray[current] + " ");

current = (current + 1) % maxSize;

count++;

}

System.out.println();

}

}

Додаток І

*Лістинг файлу «PriorityQ.java»*

class PriorityQ {

private int maxSize;

private long[] queArray;

private int front;

private int rear;

private int nItems;

public PriorityQ(int s) {

maxSize = s;

queArray = new long[maxSize];

front = 0;

rear = -1;

nItems = 0;

}

public void insert(long j) {

if (rear == maxSize - 1) {

rear = -1;

}

queArray[++rear] = j;

nItems++;

}

public long remove() {

long temp = queArray[front++];

if (front == maxSize) {

front = 0;

}

nItems--;

return temp;

}

public long peekFront() {

return queArray[front];

}

public boolean isEmpty() {

return (nItems == 0);

}

public boolean isFull() {

return (nItems == maxSize);

}

public int size() {

return nItems;

}

}

Додаток І

*Лістинг файлу «PriorityQApp.java»*

public class PriorityQApp {

public static void main(String[] args) {

PriorityQ thePQ = new PriorityQ(5);

thePQ.insert(30);

thePQ.insert(50);

thePQ.insert(10);

thePQ.insert(40);

thePQ.insert(20);

while(!thePQ.isEmpty()){

long item = thePQ.remove();

System.out.println(item + " ");

}

}

}

Додаток Ї

*Лістинг файлу «Deque.java»*

class Deque {

private int maxSize;

private long[] dequeArray;

private int left;

private int right;

private int nItems;

public Deque(int s) {

maxSize = s;

dequeArray = new long[maxSize];

left = 0;

right = -1;

nItems = 0;

}

public void displayDeque() {

if (isEmpty()) {

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

return;

}

int current = left;

int count = 0;

while (count < nItems) {

System.out.print(dequeArray[current] + " ");

current = (current + 1) % maxSize;

count++;

}

System.out.println();

}

public void insertLeft(long j) {

if (left == 0)

left = maxSize;

dequeArray[--left] = j;

nItems++;

}

public void insertRight(long j) {

if (right == maxSize - 1)

right = -1;

dequeArray[++right] = j;

nItems++;

}

public long removeLeft() {

long temp = dequeArray[left++];

if (left == maxSize)

left = 0;

nItems--;

return temp;

}

public long removeRight() {

long temp = dequeArray[right--];

if (right == -1)

right = maxSize - 1;

nItems--;

return temp;

}

public boolean isEmpty() {

return (nItems == 0);

}

public boolean isFull() {

return (nItems == maxSize);

}

}

Додаток Ж

*Лістинг файлу «DequeApp.java»*

public class DequeApp {

public static void main(String[] args) {

Deque theDeque = new Deque(5);

theDeque.insertLeft(10);

theDeque.insertRight(20);

theDeque.insertLeft(30);

theDeque.insertRight(40);

theDeque.displayDeque();

System.out.println("Removed from left: " + theDeque.removeLeft());

System.out.println("Removed from right: " + theDeque.removeRight());

theDeque.insertLeft(50);

theDeque.insertRight(60);

theDeque.displayDeque();

while (!theDeque.isEmpty()) {

System.out.println("Removed from left: " + theDeque.removeLeft());

}

theDeque.displayDeque();

}

}

Додаток З

*Лістинг файлу «StackWithDequeApp.java»*

public class StackWithDequeApp {

public static void main(String[] args) {

StackWithDeque theStack = new StackWithDeque(5);

theStack.insertRight(10);

theStack.insertRight(20);

theStack.insertRight(30);

theStack.displayDeque(); // Вивести вміст стека

System.out.println("Removed from right: " + theStack.removeRight());

System.out.println("Removed from left: " + theStack.removeLeft());

theStack.insertRight(40);

theStack.insertRight(50);

theStack.displayDeque(); // Вивести вміст стека

while (!theStack.isEmpty()) {

System.out.println("Removed from right: " + theStack.removeRight());

}

theStack.displayDeque(); // Вивести вміст стека

}

}

Додаток И

*Лістинг файлу «StackWithDeque.java»*

class StackWithDeque {

private Deque deque;

public StackWithDeque(int maxSize) {

deque = new Deque(maxSize);

}

public void insertLeft(long j) {

deque.insertLeft(j);

}

public void insertRight(long j) {

deque.insertRight(j);

}

public long removeLeft() {

return deque.removeLeft();

}

public long removeRight() {

return deque.removeRight();

}

public boolean isEmpty() {

return deque.isEmpty();

}

public boolean isFull() {

return deque.isFull();

}

public void displayDeque() {

deque.displayDeque();

}

}