1. **Write a program in Java to right rotate an array by 5 steps**

**package** assistedproject3;

**import** java.util.Arrays;

**public** **class** ArrayRotation {

**public** **static** **void** main(String[] args) {

**int**[] array = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

System.***out***.println("Original array: " + Arrays.*toString*(array));

// Right rotate the array by 5 steps

*rightRotate*(array, 5);

System.***out***.println("Rotated array: " + Arrays.*toString*(array));

}

**public** **static** **void** rightRotate(**int**[] array, **int** steps) {

**int** length = array.length;

steps = steps % length; // Adjust steps if it's greater than the array length

// Reverse the entire array

*reverseArray*(array, 0, length - 1);

// Reverse the first 'steps' elements

*reverseArray*(array, 0, steps - 1);

// Reverse the remaining elements

*reverseArray*(array, steps, length - 1);

}

**public** **static** **void** reverseArray(**int**[] array, **int** start, **int** end) {

**while** (start < end) {

**int** temp = array[start];

array[start] = array[end];

array[end] = temp;

start++;

end--;

}

}

}

**OUTPUT:**

Original array: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Rotated array: [6, 7, 8, 9, 10, 1, 2, 3, 4, 5]

**2.Write a program in Java to find the fourth smallest element in an unsorted list**

package assistedproject3;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class FourthSmallestElement {

public static void main(String[] args) {

List<Integer> list = new ArrayList<>();

list.add(10);

list.add(5);

list.add(8);

list.add(2);

list.add(7);

list.add(3);

list.add(1);

list.add(6);

list.add(9);

list.add(4);

int fourthSmallest = findFourthSmallest(list);

System.out.println("The fourth smallest element is: " + fourthSmallest);

}

public static int findFourthSmallest(List<Integer> list) {

if (list.size() < 4) {

throw new IllegalArgumentException("List size is less than 4");

}

// Sort the list in ascending order

Collections.sort(list);

// Return the fourth element

return list.get(3);

}

}

**OUTPUT:**

The fourth smallest element is: 4

**3.Write a program in Java to find the sum of n number of elements in the**

**range of L and R where 0 <= L <= R <= n-1**

**package** assistedproject3;

**import** java.util.Scanner;

**public** **class** SumInRange {

**public** **static** **void** main(String[] args) {

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Enter the number of elements (n): ");

**int** n = scanner.nextInt();

**int**[] arr = **new** **int**[n];

System.***out***.println("Enter the elements:");

**for** (**int** i = 0; i < n; i++) {

arr[i] = scanner.nextInt();

}

System.***out***.print("Enter the range start (L): ");

**int** L = scanner.nextInt();

System.***out***.print("Enter the range end (R): ");

**int** R = scanner.nextInt();

**int** sum = *findSumInRange*(arr, L, R);

System.***out***.println("Sum of elements in the range [" + L + ", " + R + "]: " + sum);

}

**public** **static** **int** findSumInRange(**int**[] arr, **int** L, **int** R) {

**if** (L < 0 || L >= arr.length || R < 0 || R >= arr.length || L > R) {

**throw** **new** IllegalArgumentException("Invalid range");

}

**int** sum = 0;

**for** (**int** i = L; i <= R; i++) {

sum += arr[i];

}

**return** sum;

}

}

**OUTPUT:**

Enter the number of elements (n): 6

Enter the elements:

2

5

4

3

9

7

Enter the range start (L): 1

Enter the range end (R): 3

Sum of elements in the range [1, 3]: 12

1. **Write a program in Java to multiply two matrices**

**package** assistedproject3;

**public** **class** MatrixMultiplication {

**public** **static** **void** main(String[] args) {

**int**[][] matrix1 = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

**int**[][] matrix2 = {

{10, 11},

{12, 13},

{14, 15}

};

**int**[][] result = *multiplyMatrices*(matrix1, matrix2);

System.***out***.println("Resultant matrix:");

*printMatrix*(result);

}

**public** **static** **int**[][] multiplyMatrices(**int**[][] matrix1, **int**[][] matrix2) {

**int** rows1 = matrix1.length;

**int** cols1 = matrix1[0].length;

**int** rows2 = matrix2.length;

**int** cols2 = matrix2[0].length;

**if** (cols1 != rows2) {

**throw** **new** IllegalArgumentException("Cannot multiply matrices. Invalid dimensions.");

}

**int**[][] result = **new** **int**[rows1][cols2];

**for** (**int** i = 0; i < rows1; i++) {

**for** (**int** j = 0; j < cols2; j++) {

**for** (**int** k = 0; k < cols1; k++) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

**return** result;

}

**public** **static** **void** printMatrix(**int**[][] matrix) {

**int** rows = matrix.length;

**int** cols = matrix[0].length;

**for** (**int** i = 0; i < rows; i++) {

**for** (**int** j = 0; j < cols; j++) {

System.***out***.print(matrix[i][j] + " ");

}

System.***out***.println();

}

}

}

**OUTPUT:**

Resultant matrix:

76 82

184 199

292 316

**5.Write a program in Java to delete the first occurrence of a key in a singly linked list**

**package** assistedproject3;

**public** **class** LinkedList {

**static** **class** Node {

**int** data;

Node next;

Node(**int** data) {

**this**.data = data;

next = **null**;

}

}

**private** Node head;

**public** LinkedList() {

head = **null**;

}

**public** **void** insert(**int** data) {

Node newNode = **new** Node(data);

**if** (head == **null**) {

head = newNode;

} **else** {

Node current = head;

**while** (current.next != **null**) {

current = current.next;

}

current.next = newNode;

}

}

**public** **void** delete(**int** key) {

**if** (head == **null**) {

System.***out***.println("List is empty. Cannot delete.");

**return**;

}

**if** (head.data == key) {

head = head.next;

**return**;

}

Node prev = **null**;

Node current = head;

**while** (current != **null** && current.data != key) {

prev = current;

current = current.next;

}

**if** (current == **null**) {

System.***out***.println("Key not found in the list.");

**return**;

}

prev.next = current.next;

}

**public** **void** display() {

**if** (head == **null**) {

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

**return**;

}

Node current = head;

**while** (current != **null**) {

System.***out***.print(current.data + " ");

current = current.next;

}

System.***out***.println();

}

**public** **static** **void** main(String[] args) {

LinkedList list = **new** LinkedList();

list.insert(10);

list.insert(20);

list.insert(30);

list.insert(40);

list.insert(50);

System.***out***.print("Original list: ");

list.display();

**int** key = 30;

list.delete(key);

System.***out***.print("List after deleting " + key + ": ");

list.display();

}

}

**OUTPUT:**

Original list: 10 20 30 40 50

List after deleting 30: 10 20 40 50

**6.Write a program in Java to insert a new element in a sorted circular linked list**

**package** assistedproject3;

**public** **class** CircularLinkedList {

**static** **class** Node {

**int** data;

Node next;

Node(**int** data) {

**this**.data = data;

next = **null**;

}

}

**private** Node head;

**public** CircularLinkedList() {

head = **null**;

}

**public** **void** insert(**int** data) {

Node newNode = **new** Node(data);

**if** (head == **null**) {

head = newNode;

head.next = head;

} **else** **if** (data <= head.data) {

Node last = getLastNode();

last.next = newNode;

newNode.next = head;

head = newNode;

} **else** {

Node current = head;

**while** (current.next != head && current.next.data < data) {

current = current.next;

}

newNode.next = current.next;

current.next = newNode;

}

}

**private** Node getLastNode() {

Node last = head;

**while** (last.next != head) {

last = last.next;

}

**return** last;

}

**public** **void** display() {

**if** (head == **null**) {

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

**return**;

}

Node current = head;

**do** {

System.***out***.print(current.data + " ");

current = current.next;

} **while** (current != head);

System.***out***.println();

}

**public** **static** **void** main(String[] args) {

CircularLinkedList list = **new** CircularLinkedList();

list.insert(10);

list.insert(20);

list.insert(30);

list.insert(40);

list.insert(50);

System.***out***.print("Original list: ");

list.display();

**int** newData = 25;

list.insert(newData);

System.***out***.print("List after inserting " + newData + ": ");

list.display();

}

}

**OUTPUT:**

Original list: 10 20 30 40 50

List after inserting 25: 10 20 25 30 40 50

**7.Write a program in Java to traverse a doubly linked list in the forward and backward directions**

**package** assistedproject3;

**public** **class** DoubleLinkedList {

**static** **class** Node {

**int** data;

Node prev;

Node next;

Node(**int** data) {

**this**.data = data;

prev = **null**;

next = **null**;

}

}

**private** Node head;

**public** DoubleLinkedList() {

head = **null**;

}

**public** **void** insert(**int** data) {

Node newNode = **new** Node(data);

**if** (head == **null**) {

head = newNode;

} **else** {

Node current = head;

**while** (current.next != **null**) {

current = current.next;

}

current.next = newNode;

newNode.prev = current;

}

}

**public** **void** traverseForward() {

**if** (head == **null**) {

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

**return**;

}

Node current = head;

**while** (current != **null**) {

System.***out***.print(current.data + " ");

current = current.next;

}

System.***out***.println();

}

**public** **void** traverseBackward() {

**if** (head == **null**) {

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

**return**;

}

Node current = head;

**while** (current.next != **null**) {

current = current.next;

}

**while** (current != **null**) {

System.***out***.print(current.data + " ");

current = current.prev;

}

System.***out***.println();

}

**public** **static** **void** main(String[] args) {

DoubleLinkedList list = **new** DoubleLinkedList();

list.insert(10);

list.insert(20);

list.insert(30);

list.insert(40);

list.insert(50);

System.***out***.print("Forward traversal: ");

list.traverseForward();

System.***out***.print("Backward traversal: ");

list.traverseBackward();

}

}

**OUTPUT:**

Forward traversal: 10 20 30 40 50

Backward traversal: 50 40 30 20 10

**8.Write a program in Java to insert and remove elements in a stack**

**package** assistedproject3;

**public** **class** Stack {

// Stack implementation in Java

// store elements of stack

**private** **int** arr[];

// represent top of stack

**private** **int** top;

// total capacity of the stack

**private** **int** capacity;

// Creating a stack

Stack(**int** size) {

// initialize the array

// initialize the stack variables

arr = **new** **int**[size];

capacity = size;

top = -1;

}

// push elements to the top of stack

**public** **void** push(**int** x) {

**if** (isFull()) {

System.***out***.println("Stack OverFlow");

// terminates the program

System.*exit*(1);

}

// insert element on top of stack

System.***out***.println("Inserting " + x);

arr[++top] = x;

}

// pop elements from top of stack

**public** **int** pop() {

// if stack is empty

// no element to pop

**if** (isEmpty()) {

System.***out***.println("STACK EMPTY");

// terminates the program

System.*exit*(1);

}

// pop element from top of stack

**return** arr[top--];

}

// return size of the stack

**public** **int** getSize() {

**return** top + 1;

}

// check if the stack is empty

**public** Boolean isEmpty() {

**return** top == -1;

}

// check if the stack is full

**public** Boolean isFull() {

**return** top == capacity - 1;

}

// display elements of stack

**public** **void** printStack() {

**for** (**int** i = 0; i <= top; i++) {

System.***out***.print(arr[i] + ", ");

}

}

**public** **static** **void** main(String[] args) {

Stack stack = **new** Stack(5);

stack.push(1);

stack.push(2);

stack.push(3);

System.***out***.print("Stack: ");

stack.printStack();

// remove element from stack

stack.pop();

System.***out***.println("\nAfter popping out");

stack.printStack();

}

}

**OUTPUT:**

Inserting 1

Inserting 2

Inserting 3

Stack: 1, 2, 3,

After popping out

1, 2,

1. **Write a program in Java to insert and remove elements in a queue**

**package** assistedproject3;

**public** **class** Queue {

**int** SIZE = 5;

**int** items[] = **new** **int**[SIZE];

**int** front, rear;

Queue() {

front = -1;

rear = -1;

}

// check if the queue is full

**boolean** isFull() {

**if** (front == 0 && rear == SIZE - 1) {

**return** **true**;

}

**return** **false**;

}

// check if the queue is empty

**boolean** isEmpty() {

**if** (front == -1)

**return** **true**;

**else**

**return** **false**;

}

// insert elements to the queue

**void** enQueue(**int** element) {

// if queue is full

**if** (isFull()) {

System.***out***.println("Queue is full");

}

**else** {

**if** (front == -1) {

// mark front denote first element of queue

front = 0;

}

rear++;

// insert element at the rear

items[rear] = element;

System.***out***.println("Insert " + element);

}

}

// delete element from the queue

**int** deQueue() {

**int** element;

// if queue is empty

**if** (isEmpty()) {

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

**return** (-1);

}

**else** {

// remove element from the front of queue

element = items[front];

// if the queue has only one element

**if** (front >= rear) {

front = -1;

rear = -1;

}

**else** {

// mark next element as the front

front++;

}

System.***out***.println( element + " Deleted");

**return** (element);

}

}

// display element of the queue

**void** display() {

**int** i;

**if** (isEmpty()) {

System.***out***.println("Empty Queue");

}

**else** {

// display the front of the queue

System.***out***.println("\nFront index-> " + front);

// display element of the queue

System.***out***.println("Items -> ");

**for** (i = front; i <= rear; i++)

System.***out***.print(items[i] + " ");

// display the rear of the queue

System.***out***.println("\nRear index-> " + rear);

}

}

**public** **static** **void** main(String[] args) {

// create an object of Queue class

Queue q = **new** Queue();

q.deQueue();

// insert elements to the queue

**for**(**int** i = 1; i < 6; i ++) {

q.enQueue(i);

}

// 6th element can't be added to queue because queue is full

q.enQueue(6);

q.display();

// deQueue removes element entered first i.e. 1

q.deQueue();

// Now we have just 4 elements

q.display();

}

}

**OUTPUT:**

Queue is empty

Insert 1

Insert 2

Insert 3

Insert 4

Insert 5

Queue is full

Front index-> 0

Items ->

1 2 3 4 5

Rear index-> 4

1 Deleted

Front index-> 1

Items ->

2 3 4 5

Rear index-> 4