**public class SelectionSort {**

public static void main(String[] args) {

int[] arr = {64, 25, 12, 22, 11};

System.out.println("Original Array:");

printArray(arr);

selectionSort(arr);

System.out.println("\nSorted Array:");

printArray(arr);

}

public static void selectionSort(int[] arr) {

int n = arr.length;

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

for (int j = i + 1; j < n; j++) {

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

public static void printArray(int[] arr) {

for (int num : arr) {

System.out.print(num + " ");

}

System.out.println();

}

}

**public class InsertionSort {**

public static void main(String[] args) {

int[] arr = {64, 25, 12, 22, 11}; // Example array to be sorted

System.out.println("Original Array:");

printArray(arr);

insertionSort(arr); // Perform insertion sort

System.out.println("\nSorted Array:");

printArray(arr);

}

// Insertion Sort function

public static void insertionSort(int[] arr) {

int n = arr.length;

// Loop through elements starting from the second element

for (int i = 1; i < n; i++) {

int key = arr[i]; // The current element to be inserted into the sorted part

int j = i - 1;

// Shift elements of the sorted part that are greater than the key to one position ahead

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

// Place the key in the correct position

arr[j + 1] = key;

}

}

// Function to print the array

public static void printArray(int[] arr) {

for (int num : arr) {

System.out.print(num + " ");

}

System.out.println();

}

}

public class InsertionSort {

public static void main(String[] args) {

int[] arr = {64, 25, 12, 22, 11}; // Example array to be sorted

System.out.println("Original Array:");

printArray(arr);

insertionSort(arr); // Perform insertion sort

System.out.println("\nSorted Array:");

printArray(arr);

}

// Insertion Sort function

public static void insertionSort(int[] arr) {

int n = arr.length;

// Loop through elements starting from the second element

for (int i = 1; i < n; i++) {

int key = arr[i]; // The current element to be inserted into the sorted part

int j = i - 1;

// Shift elements of the sorted part that are greater than the key to one position ahead

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j--;

}

// Place the key in the correct position

arr[j + 1] = key;

}

}

// Function to print the array

public static void printArray(int[] arr) {

for (int num : arr) {

System.out.print(num + " ");

}

System.out.println();

}

}

**class Stack {**

    private int[] stackArray;

    private int top;

    private int capacity;

    public Task1(int initialCapacity) {

        stackArray = new int[initialCapacity];

        capacity = initialCapacity;

        top = -1;

    }

    public void push(int value) {

        if (top == capacity - 1) {

            increaseCapacity();

        }

        stackArray[++top] = value;

        System.out.println("Pushed " + value + " to stack.");

    }

    public int pop() {

        if (isEmpty()) {

            System.out.println("Stack Underflow! Cannot pop.");

            return -1;

        }

        return stackArray[top--];

    }

    public int peek() {

        if (isEmpty()) {

            System.out.println("Stack is empty! Nothing to peek.");

            return -1;

        }

        return stackArray[top];

    }

    public boolean isEmpty() {

        return top == -1;

    }

    public int size() {

        return top + 1;

    }

    public void display() {

        if (isEmpty()) {

            System.out.println("Stack is empty!");

            return;

        }

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

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

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

        }

        System.out.println();

    }

    private void increaseCapacity() {

        int newCapacity = capacity \* 2;

        int[] newArray = new int[newCapacity];

        System.arraycopy(stackArray, 0, newArray, 0, capacity);

        stackArray = newArray;

        capacity = newCapacity;

        System.out.println("Stack capacity increased to " + newCapacity);

    }

    public static void main(String[] args) {

        Task1 stack = new Task1(3);

        stack.push(10);

        stack.push(20);

        stack.push(30);

        stack.push(40);

        System.out.println("Top element is " + stack.peek());

        stack.display();

        System.out.println("Popped element is " + stack.pop());

        System.out.println("Current stack size is " + stack.size());

        stack.display();

        System.out.println("Is stack empty? " + stack.isEmpty());

    }

}

**public class Queue {**

    private int[] arr;

    private int front;

    private int rear;

    private int size;

    public Task1(int capacity) {

        arr = new int[capacity];

        front = 0;

        rear = -1;

        size = 0;

    }

    public void clear() {

        front = 0;

        rear = -1;

        size = 0;

    }

    public boolean isEmpty() {

        return size == 0;

    }

    public void enqueue(int el) {

        if (size == arr.length) {

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

        } else {

            rear = (rear + 1) % arr.length;

            arr[rear] = el;

            size++;

        }

    }

    public int dequeue() {

        if (isEmpty()) {

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

            return -1;

        } else {

            int removedElement = arr[front];

            front = (front + 1) % arr.length;

            size--;

            return removedElement;

        }

    }

    public int firstEl() {

        if (isEmpty()) {

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

            return -1;

        } else {

            return arr[front];

        }

    }

    public int lastEl() {

        if (isEmpty()) {

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

            return -1;

        } else {

            return arr[rear];

        }

    }

    public void display() {

        if (isEmpty()) {

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

        } else {

            System.out.print("Queue elements: ");

            for (int i = 0; i < size; i++) {

                System.out.print(arr[(front + i) % arr.length] + " ");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        Task1 queue = new Task1(5);

        queue.enqueue(10);

        queue.enqueue(20);

        queue.enqueue(30);

        queue.enqueue(40);

        queue.enqueue(50);

        queue.enqueue(60);

        queue.enqueue(70);

        queue.display();

        System.out.println("First element: " + queue.firstEl());

        System.out.println("Last element: " + queue.lastEl());

        queue.dequeue();

        queue.display();

        queue.clear();

        System.out.println("Queue cleared");

        queue.display();

    }

}

**public class DoubleendedQueue {**

    private int[] arr;

    private int front;

    private int rear;

    private int size;

    public Task2(int capacity) {

        arr = new int[capacity];

        front = -1;

        rear = 0;

        size = 0;

    }

    public boolean isEmpty() {

        return size == 0;

    }

    public boolean isFull() {

        return size == arr.length;

    }

    public void insertFront(int el) {

        if (isFull()) {

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

        } else {

            front = (front - 1 + arr.length) % arr.length;

            arr[front] = el;

            size++;

        }

    }

    public void insertRear(int el) {

        if (isFull()) {

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

        } else {

            arr[rear] = el;

            rear = (rear + 1) % arr.length;

            size++;

        }

    }

    public int deleteFront() {

        if (isEmpty()) {

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

            return -1;

        } else {

            int removedElement = arr[front];

            front = (front + 1) % arr.length;

            size--;

            return removedElement;

        }

    }

    public int deleteRear() {

        if (isEmpty()) {

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

            return -1;

        } else {

            rear = (rear - 1 + arr.length) % arr.length;

            int removedElement = arr[rear];

            size--;

            return removedElement;

        }

    }

    public int getFront() {

        if (isEmpty()) {

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

            return -1;

        } else {

            return arr[front];

        }

    }

    public int getRear() {

        if (isEmpty()) {

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

            return -1;

        } else {

            return arr[(rear - 1 + arr.length) % arr.length];

        }

    }

    public void clear() {

        front = -1;

        rear = 0;

        size = 0;

    }

    public void display() {

        if (isEmpty()) {

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

        } else {

            System.out.print("Deque elements: ");

            for (int i = 0; i < size; i++) {

                System.out.print(arr[(front + i) % arr.length] + " ");

            }

            System.out.println();

        }

    }

    public static void main(String[] args) {

        Task2 deque = new Task2(5);

        deque.insertRear(10);

        deque.insertRear(20);

        deque.insertFront(5);

        deque.insertFront(3);

        deque.display();

        System.out.println("Front element: " + deque.getFront());

        System.out.println("Rear element: " + deque.getRear());

        deque.deleteFront();

        deque.deleteRear();

        deque.display();

        deque.clear();

        System.out.println("Deque cleared");

        deque.display();

    }

}

**#######BINARY TREE**  
  
class Node {

    int data;

    Node left, right;

    public Node(int item) {

        data = item;

        left = right = null;

    }

}

public class Task1 {

    Node root;

    public void insert(int data) {

        root = insertRec(root, data);

    }

    private Node insertRec(Node root, int data) {

        if (root == null) {

            root = new Node(data);

            return root;

        }

        if (data < root.data) {

            root.left = insertRec(root.left, data);

        } else if (data > root.data) {

            root.right = insertRec(root.right, data);

        }

        return root;

    }

    public void inorder() {

        inorderRec(root);

    }

    private void inorderRec(Node root) {

        if (root != null) {

            inorderRec(root.left);

            System.out.print(root.data + " ");

            inorderRec(root.right);

        }

    }

    public static void main(String[] args) {

        Task1 tree = new Task1();

        tree.insert(50);

        tree.insert(30);

        tree.insert(20);

        tree.insert(40);

        tree.insert(70);

        tree.insert(60);

        tree.insert(80);

        System.out.println("In-order traversal of the binary tree:");

        tree.inorder();

    }

}

**##########BST##########**

class Node {

    int data;

    Node left, right;

    public Node(int item) {

        data = item;

        left = right = null;

    }

}

public class Task2 {

    Node root;

    public void insert(int data) {

        root = insertRec(root, data);

    }

    private Node insertRec(Node root, int data) {

        if (root == null) {

            root = new Node(data);

            return root;

        }

    if (data < root.data) {

            root.left = insertRec(root.left, data);

        } else if (data > root.data) {

            root.right = insertRec(root.right, data);

        }

        return root;

    }

    public void inorder() {

        inorderRec(root);

    }

    private void inorderRec(Node root) {

        if (root != null) {

            inorderRec(root.left);

            System.out.print(root.data + " ");

            inorderRec(root.right);

        }

    }

    public static void main(String[] args) {

        Task2 bst = new Task2();

        bst.insert(50);

        bst.insert(30);

        bst.insert(20);

        bst.insert(40);

        bst.insert(70);

        bst.insert(60);

        bst.insert(80);

        System.out.println("In-order traversal of the binary search tree:");

        bst.inorder();

    }

}