1. Implement ‘Stack’ data structures using integer array

```
package dsa;

import java.util.Scanner;

public class Stack {
	int top;
	int maxSize = 10;
	int[] arr = new int[maxSize];

	public Stack() {
		top = -1;
	}

	public void push(Scanner sc) {
		if (this.top == this.maxSize - 1) {
			System.out.println("Stack is full");
		} else {
			System.out.println("Enter the value : ");
			int val = sc.nextInt();
			arr[++top] = val;
			System.out.println(val + " is pushed");
		}
	}

	public void pop() {
		if (top == -1) {
			System.out.println("Stack is Empty");
		} else {
			System.out.println(arr[top--] + " is popped");
		}
	}

	public void display() {
		System.out.println("Printing stack elements .....");
		for (int i = top; i >= 0; i--) {
			System.out.println(arr[i]);
		}
	}

	public static void main(String[] args) {
		int choice = 0;
		Scanner sc = new Scanner(System.in);
		Stack s = new Stack();
		System.out.println("*********Stack operations using array*********\n");
		System.out.println("\n-------------------------------------------------\n");

		while (choice != 4) {
			System.out.println("\nChose one from the below options...\n");
			System.out.println("\n1.Push\n2.Pop\n3.Show\n4.Exit");
			System.out.println("\n Enter your choice \n");
			choice = sc.nextInt();
			switch (choice) {
			case 1: {
				s.push(sc);
				break;
			}
			case 2: {
				s.pop();
				break;
			}
			case 3: {
				s.display();
				break;
			}
			case 4: {
				System.out.println("Exiting....");
				System.exit(0);
				break;
			}
			default: {
				System.out.println("Please Enter valid choice ");
			}
			}
			;}
		}
	}
```

2. Implement ‘Queue’ data structures using integer array


```
package dsa;

public class Queue {

	private static int front, rear, capacity;
	private static int queue[];

	Queue(int c) {
		front = rear = 0;
		capacity = c;
		queue = new int[capacity];
	}

	void queueEnqueue(int data) {
		// check queue is full or not
		if (capacity == rear) {
			System.out.printf("\nQueue is full\n");
			return;
		}
		// insert element at the rear
		else {
			queue[rear] = data;
			rear++;
		}
		return;
	}

	void queueDequeue() {
		// if queue is empty
		if (front == rear) {
			System.out.printf("\nQueue is empty\n");
			return;
		}
		// shift all the elements from index 2 till rear
		// to the right by one
		else {
			for (int i = 0; i < rear - 1; i++) {
				queue[i] = queue[i + 1];
			}
			// store 0 at rear indicating there's no element
			if (rear < capacity)
				queue[rear] = 0;
			// decrement rear
			rear--;
		}
		return;
	}

	void queueDisplay() {
		int i;
		if (front == rear) {
			System.out.printf("\nQueue is Empty\n");
			return;
		}
		// traverse front to rear and print elements
		for (i = front; i < rear; i++) {
			System.out.printf(" %d <-- ", queue[i]);
		}
		return;
	}
	void queueFront() {
		if (front == rear) {
			System.out.printf("\nQueue is Empty\n");
			return;
		}
		System.out.printf("\nFront Element is: %d", queue[front]);
		return;
	}

	public static void main(String[] args) {
		Queue q = new Queue(4);
		// print Queue elements
		q.queueDisplay();
		// inserting elements in the queue
		q.queueEnqueue(20);
		q.queueEnqueue(30);
		q.queueEnqueue(40);
		q.queueEnqueue(50);
		// print Queue elements
		q.queueDisplay();
		// insert element in the queue
		q.queueEnqueue(60);
		// print Queue elements
		q.queueDisplay();
		q.queueDequeue();
		q.queueDequeue();
		System.out.printf("\n\nafter two node deletion\n\n");
		// print Queue elements
		q.queueDisplay();
		// print front of the queue
		q.queueFront();
	}
}
```