큐 는 먼저 들어온 데이터가 먼저 나가는 특징을 가지고 있는 선형 자료구조이다.

큐의 특징

1. 선입선출 : 먼저 들어온 데이터가 먼저 나간다.
2. 삽입 : 큐의 뒷부분에 데이터를 추가한다.
3. 삭제 : 큐의 앞부분에서 데이터를 삭제한다.
4. 단방향 접근 : 일반적으로 큐의 앞부분과 뒷부분에서만 데이터에 접근할 수 있다.

자바에서 큐 (List) 선언시

```java
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<>();
        //데이터 추가
        queue.offer(11);
        queue.offer(22);
        queue.offer(33);
        //데이터 확인 및 삭제
        while (!queue.isEmpty()) {
            System.out.println(queue.peek()); //맨 앞 데이터 확인
            int data = queue.poll(); //맨 앞 데이터 삭제 및 반환
            System.out.println(data);
        }
    }
}
```

offer : 큐의 뒷부분에 데이터를 추가
poll : 큐의 앞부분에서 데이터를 삭제하고 반환 만약 큐가 비어있으면 null을 반환
peek : 큐의 앞부분에 있는 데이터를 확인 큐를 변경하지 않고 데이터만 확인
isEmpty : 큐가 비어있는지 확인

자바 코드 구현 (원형 큐)

원형 큐는 배열의 끝에 도달하면 다시 처음으로 돌아가서 데이터를 저장하는 방식

```java
public class MyQueue {
    private int[] arr;
    private int front;
    private int rear;
```

```
    private int capacity;

    public MyQueue(int size) {
        arr = new int[size];
        front = 0;
        rear = -1;
        capacity = size;
    }
    //큐가 비어있는지 확인
    public boolean isEmpty() {
        return front == rear + 1;
    }
    //큐가 가득 찼는지 확인
    public boolean isFull() {
        return rear == capacity - 1;
    }
    //큐에 데이터 추가
    public void enqueue(int data) {
        if (isFull()) {
            System.out.println("풀");
            return;
        }
        rear = (rear + 1) % capacity;
        arr[rear] = data;
        System.out.println(data + "추가");
    }
    //큐에서 데이터 삭제
    public int dequeue() {
        if (isEmpty()) {
            System.out.println("삭제);
            return -1;
        }
        int data = arr[front];
        front = (front + 1) % capacity;
        return data;
    }
    //큐의 front 값 확인
    public int peek() {
        if (isEmpty()) {
            return -1;
        }
        return arr[front];
    }
    public static void main(String[] args) {
        MyQueue q = new MyQueue(5);
        q.enqueue(11);
```

```
        q.enqueue(22);
        q.enqueue(33);
        System.out.println(q.dequeue());
        System.out.println(q.peek());
    }
}
```

arr : 큐에 저장될 데이터를 저장하는 배열
front : 큐의 앞쪽을 가리키는 인덱스
rear : 큐의 뒤쪽을 가리키는 인덱스
capacity : 큐의 최대 크기
enqueue : 큐에 데이터를 추가하는 메소드
dequeue: 큐에서 데이터를 삭제하는 메소드
peek : 큐의 앞쪽 데이터를 확인하는 메소드
isEmpty : 큐가 비어있는지 확인하는 메소드
isFull : 큐가 가득 찼는지 확인하는 메소드

자바 코드 구현 (덱 큐)

```
import java.util.ArrayDeque;
import java.util.Deque;

public class DequeExample {
    public static void main(String[] args) {
        Deque<Integer> deque = new ArrayDeque<>();
        //데이터 삽입
        deque.addLast(10); // 덱의 끝에 10 추가
        deque.addFirst(5);  // 덱의 앞에 5 추가

        //데이터 추출
        int firstElement = deque.removeFirst(); //덱의 앞에서 5 제거
        int lastElement = deque.removeLast();  //덱의 끝에서 10 제거

        System.out.println(firstElement); //출력: 5
        System.out.println(lastElement);  //출력: 10
    }
}
```

addFirst(e) : 덱의 앞에 요소 e를 추가
addLast(e) : 덱의 끝에 요소 e를 추가
removeFirst() : 덱의 앞에서 요소를 제거하고 반환
removeLast() : 덱의 끝에서 요소를 제거하고 반환

peekFirst() : 덱의 앞에 있는 요소를 반환하지만 제거하지 않음
peekLast() : 덱의 끝에 있는 요소를 반환하지만 제거하지 않음

Queue vs 배열 vs 덱

특징	큐	배열	덱(Deque)
정의	먼저 들어온 데이터가 먼저 나가는 선입선출(FIFO) 자료구조	동일한 자료형의 데이터를 연속된 메모리 공간에 저장하는 자료구조	양쪽 끝에서 데이터를 삽입하거나 삭제할 수 있는 자료구조
데이터 삽입/삭제	한쪽 끝	어디든 가능	양쪽 끝
랜덤 접근	느림	빠름	빠름 (ArrayDeque)
메모리 사용	노드 연결 (LinkedList) 또는 배열 (ArrayDeque)	연속된 메모리	노드 연결 또는 배열
유연성	낮음	낮음	높음
활용	FIFO, 대기열	데이터 저장, 배열 연산	스택, 큐, 양방향 큐