**1. 큐(queue)란**

- 'FIFO 리스트'이다.

- 앞(front)에서 삭제(dequeue)가, 뒤(rear)에서 삽입(enqueue)이 이루어진다.

- 언더플로우(underflow) : 빈 큐에서 dequeue을 시도할 경우 발생, 예외처리 필요

- 오버플로우(overflow) : 꽉 찬 큐에서 enqueue를 시도할 경우 발생, 예외처리 필요

**2. 큐 ADT**

**1) 주요연산ADT**

- void enQueue(Object data);

- Object deQueue(); //자바에서는 poll();

**2) 보조연산ADT**

- Object peek();   // front에 있는 항목 리턴. poll()과 비교했을 때 조회만 한다.

- int size();   // 항목 갯수 리턴

- Object isEmpty();   // dequeue()에서 호출, underflow여부

- Object isFull();   // enqueue()에서 호출, overflow여부

**3. 큐 적용사례**

- (우선순위 같은) 시스템 예약작업 도착순서대로 처리하기

- 멀티프로그래밍

- 비동기적 데이터전송(파일입출력, 파이프, 소켓)

- '대기 줄'에 비유되는 실제상황

**4. '원형 큐'의 구현**

​- 원형 큐에서는 처음에 front = rear = 0으로 초기화한다.

- 원형 큐에서는 front와 rear 사이에 공백이 존재하므로 실제크기-1만큼 사용할 수 있다.

**1) 배열구현**

**(1) 정적 원형배열**

​public class ArrayQueue {

private int front, rear;

private int[] arr;

public ArrayQueue(int size) {

front = rear = 0;

arr = new int[size+1]; //필요한 크기+1만큼 할당했다.

}

public boolean isEmpty() {

return (front == rear);

}

public boolean isFull() {

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

}

public boolean enqueue(int data) {

if(isFull()) {

throw new QueueOverflowException("Queue Overflow");

return false;

}

arr[rear] = data;

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

return true;

}

public int dequeue() {

if(isEmpty()) {

throw new EmptyQueueException("Queue Empty");

return -1;

}

int data = arr[front];

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

return data;

}

public int getQueueSize() {

return ((arr.length + (rear - front))%capacity);

}

}

**(2) 동적 원형배열**

​public class DynArrayQueue {

private int front, rear;

private int[] arr;

public DynArrayQueue() {

front = rear = 0;

arr = new int[1]; // 크기고민이 필요없다.

}

public boolean isEmpty() {

return (front == rear);

}

public boolean isFull() {

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

}

public boolean enqueue(int data) {

if(isFull()) resizeQueue();

arr[rear] = data;

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

return true;

}

public int dequeue() {

if(isEmpty()) {

throw new EmptyQueueException("Queue Empty");

return -1;

}

int data = arr[front];

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

return data;

}

**private boolean resizeQueue() { //이거 제대로 될까? 테스트 안해봄.**

**int[] oldArr = arr;**

**arr = new int[arr.length\*2];**

**for(int i=0; !isEmpty(); i++) {**

**arr[i] = oldArr[(front+i)%oldArr.length];**

**}**

**front=0;**

**rear=arr.length;**

**return true;**

**}**

}

**(3) 연결리스트 구현**

public class ListQueue {

private ListNode frontNode;

private ListNode rearNode;

public ListNode() {

frontNode = rearNode = null;

}

public boolean isEmpty() {

return (frontNode == null);

}

//isFull() 필요없다.

**public void enqueue(int data) { //순서는 상관없겠지?**

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

**if(frontNode == null) {**

**frontNode = rearNode;**

**}**

**if(rearNode != null) {**

**rearNode.setNext(rearNode);**

**}**

**rearNode = newNode;**

**}**

**public int dequeue() {**

**int data;**

**if(isEmpty()) throw new EmptyQueueException("Queue Empty");**

**data = frontNode.getData();**

**frontNode = frontNode.getNext();**

**return data;**

**}**

}