자료구조와 함께 배우는 알고리즘 입문(자바편)_4장_스택과 큐

1. 스택(Stack)

- 데이터를 일시적으로 쌓아 놓는 자료구조
- 후입선출(LIFO, Last In First Out)
- 기본 용어
  1. push: 푸시(스택에 데이터 넣기)
  2. pop: 팝(스택에서 데이터 꺼내기)
  3. peek: 피크(스택의 꼭대기에 있는 데이터 들여다보기)
  4. top: 푸시와 팝이 이루어지는 쪽
  5. bottom: 스택의 가장 아랫부분
  6. 용량: 스택에 쌓을 수 있는 최대 데이터 수

- int형 고정 길이 스택 만들기

```
public class IntStact
{
  private int[] stk;     // 스택용 배열(푸시된 데이터를 저장하는 스택용 배열)
  private int capacity;  // 스택 용량(스택에 쌓을 수 있는 최대 데이터 수)
  private int ptr;       // 스택 포인터(스택에 쌓여 있는 데이터 수를 나타내는 필드)

  // 실행 시 예외: 스택이 비어 있음
  public class EmptyIntStackException extends RuntimeException
  {
    public EmptyIntStackException()
    {}
  }

  // 실행 시 예외: 스택이 가득 참
  public class OverflowIntStackException extends RuntimeException
  {
    public OverflowIntStackException()
    {}
  }

  // 생성자(스택용 배열 본체 생성 등 준비작업)
  public IntStack(int maxlen)
  {
```

```
    ptr = 0;                        // 데이터가 하나도 쌓여 있지 않은 상태
    capacity = maxlen;
    try
    {
      stk = new int[capacity];      // 스택 본체용 배열을 생성
    }
    catch (OutOfMemoryError e)      // 생성할 수 없음(배열 본체 생성 실패)
    {
      capacity = 0;                 // 존재하지 않는 배열 stk에 다른 메서드의 접근 제어
    }
  }

  // 스택에 x를 푸시
  public int push(int x) throws OverflowIntStackException
  {
    if (ptr >= capacity)            // 스택이 가득 참
      throw new OverflowIntStackException();
    return stk[ptr++] = x;
  }

  // 스택에서 데이터를 팝(꼭대기에 있는 데이터를 꺼냄)
  public int pop() throws EmptyIntStackException
  {
    if (ptr <= 0)                   // 스택이 비어 있음
      throw new EmptyIntStackException();
    return stk[--ptr];
  }

  // 스택에서 데이터를 피크(꼭대기에 있는 데이터를 들여다봄)
  public int peek() throws EmptyIntStackException
  {
    if (ptr <= 0)                   // 스택이 비어 있음
      throw new EmptyIntStackException();
    return stk[ptr -1];
  }

  // 스택을 비움
  public void clear()
  {
    ptr = 0;
  }

  // 스택에서 x를 찾아 인덱스(없으면 -1)를 반환
  public int indexOf(int x)
  {
    for (int i = ptr-1; i >= 0; i--)     // 꼭대기 쪽부터 선형 검색
      if (stk[i] == x)
        return i;                        // 검색 성공
      return -1;                         // 검색 실패
  }

  // 스택의 용량을 반환
  public int getCapacity()
  {
    return capacity
  }

  // 스택에 쌓여 있는 데이터 개수를 반환
```

```
  public int size()
  {
    return ptr;
  }

  // 스택이 비어 있는지 검사
  public boolean isEmpty()
  {
    return ptr <= 0;         // ptr == 0 도 가능
  }

  // 스택이 가득 찼는지 검사
  public boolean isFull()
  {
    return ptr >= capacity;  // ptr == capacity
  }

  // 스택 안의 모든 데이터 출력(바닥 → 꼭대기)
  public void dump()
  {
    if (ptr <= 0)
      System.out.println("스택이 비어 있습니다.");
    else
    {
      for (int i = 0; i < ptr; i++)
        System.out.println(stk[i] + " ");
      System.out.println();
    }
  }
}
```

- 스택용 배열 본체의 개별 요소에 접근하는 인덱스식: stk[0], stk[1], … , stk[capacity-1]
- 스택이 가득 찼는지 판단할 때 등가 연산자(==) 사용

```
if (ptr == capacity)     // 스택이 가득 찼는가?
if (ptr == 0)            // 스택이 비어 있는가?
```

2. 큐(Queue)

- 스택과 마찬가지로 데이터를 일시적으로 쌓아 두기 위한 자료구조
- 선입선출(FIFO, First In First Out / LILO, Last In Last Out)
- 실생활에서 많이 보는 예) 은행 창구 차례, 마트 계산 대기열 등
- 기본 용어
  1. enqueue: 인큐(큐에 데이터 넣기) → O(1)

2. dequeue: 디큐(큐에서 데이터 꺼내기 → O(n) : 맨 앞 데이터 꺼내고, 나머지 데이터 모두 한 칸씩 앞으로 이동
3. front: 데이터를 꺼내는 쪽
4. rear: 데이터를 넣는 쪽

- 링 버퍼(ring buffer) 사용 Queue: dequeue 하고도 배열 요소를 앞으로 옮기지 않음
  - 처음과 끝이 연결되어 있음
  - enqueue(), dequeue() 모두 O(1)
  - **주의할 점
    1. enqueue하고 rear가 max와 같아지면 rear = 0; 처리 (순환해야하므로)
    2. dequeue하고 front가 max와 같아지면 front = 0; 처리
    3. indexOf 할 때, 인덱스의 계산 『(i + front) % max』
    4. dump할 때에도, 인덱스 계산 『(i + front) % max』

- 링 버퍼로 큐 만들어보기

```
public class RingBufferQueue
{
  private int max; // 큐의 용량
  private int front; // 첫 번째 요소 커서
  private int rear; // 마지막 요소 커서
  //-- 다음 인큐할 인덱스 미리 준비해 두는 것
  private int num; // 현재 데이터 수
  //-- front == rear 경우, 큐가 빈 건지/가득 찬 건지 구별할 수 없는 상황을 위해 필요
  private int[] que; // 큐 본체

  // 실행 시 예외: 큐가 비어 있음
  public class EmptyIntQueueException extends RuntimeException
  {
    public EmptyIntQueueException() {}
  }

  // 실행 시 예외: 큐가 가득 참
  public class OverflowIntQueueException extends RuntimeException
  {
    public OverflowIntQueueException() {}
  }

  // 생성자
  public ringBufferQueue(int capacity)
  {
    num = front = rear = 0;
    max = capacity;
```

```
    try
    {
      que = new int[max];
    }
    catch (OutOfMemoryError e)
    {
      max = 0;
    }
  }

  // enque() : 큐의 맨 뒤에 데이터 넣음
  public int enque(int x) throws OverflowIntQueueException
  {
    if (num >= max)
      throw new OverflowIntQueueException();
    que[rear++] = x;
    num++;
    if (rear == max) // *rear가 최대 용량의 max와 같을 경우, rear = 0으로 설정
      rear = 0;
    return x;
  }

  // deque() : 큐의 맨 앞 데이터 꺼냄
  public int deque() throws EmptyIntQueueException
  {
    if (num <= 0)
      throw new EmptyIntQueueException();
    int x = que[front++];
    num--;
    if (front == max) // *front가 max와 같을 경우, front = 0으로 설정
      front = 0;
    return x;
  }

  // peek() : 큐의 맨 앞 데이터 확인 (꺼내지는 X → front, rear, num 값 변화 없음)
  public int peek() throws EmptyIntQueueException
  {
    if (num <= 0)
      throw new EmptyIntQueueException();
    return que[front];
  }

  // indexOf() : 검색 값의 인덱스 반환 (없으면 -1 반환)
  public int indexOf(int x)
  {
    // front → rear 선형 검색
    for (int i = 0; i < num; i++)
    {
      int idx = (i + front) % max; // ★
      if (que[idx] == x)
        return idx;
    }
    return -1;
  }

  // search() : 검색 값의 큐 안에서의 위치 반환 (front면 1, 없으면 0)
  public int search(int x)
  {
```

```
    for (int i = 0; i < num; i++)
      if (que[(i + front) % max] == x)
        return i + 1;
    return 0;
  }

  // clear() : 큐의 모든 데이터 삭제
  public void clear()
  {
    num = front = rear = 0; // → 큐의 요솟값 바꿀 필요 X
  }

  // capacity() : 최대 용량 확인
  public int capacity()
  {
    return max;
  }

  // size() : 현재 데이터 개수 확인
  public int size()
  {
    return num;
  }

  // isEmpty() : 큐 비어 있음? (true/false 반환)
  public boolean isEmpty()
  {
    return num <= 0;
  }

  // isFull() : 큐 가득 참? (true/false 반환)
  public boolean isFull()
  {
    return num >= max;
  }

  // dump() : 큐의 모든 데이터 front → rear 순으로 출력
  public void dump()
  {
    if (num <= 0)
      System.out.println("큐 비어있음");
    else
    {
      for (int i = 0; i < num; i++)
        System.out.print(que[(i + front) % max] + " ");
      System.out.println();
    }
  }
}
```

- 링 버퍼: 오래된 데이터를 버리는 용도로도 사용할 수 있음

ex) 용량 10으로 해놓고 enqueue는 무한히 가능하게

> 데이터 10개 초과 시, 오래된 데이터는 새로 입력된 데이터로 덮어쓰기

> 가장 최근에 입력한 10개의 데이터만 링 버퍼에 남게 된다.

- 덱(deck): 양방향 대기열(deque/double ended queue), 시작과 끝 지점에서 양쪽으로 데이터를 인큐, 디큐하는 자료구조

- 덱 만들어보기

```
public class Deck
{
  private int max;    // 덱(deck)의 용량
  private int num;    // 현재 데이터 개수
  private int front;  // 맨 앞 커서
  private int rear;   // 맨 뒤 커서
  private int[] que;  // 덱(deck)의 본체

  // 실행 시 예외: deck 비어 있음
  public class EmptyDeckException extends RuntimeException
  {
      public EmptyDeckException() {}
  }

  // 실행 시 예외: deck 가득 참
  public class OverflowDeckException extends RuntimeException
  {
      public OverflowDeckException() {}
  }

  // 생성자
  public deck(int capacity)
  {
      num = front = rear = 0;
      max = capacity;
      try
      {
          que = new int[max];
      }
      catch (OutOfMemoryError e)
      {
          max = 0;
      }
  }

  // enqueFront() : 데이터를 front에 enque
  public int enqueFront(int x) throws OverflowDeckException
  {
      if (num >= max)
          throw new OverflowDeckException();
      if (--front < 0)
          front = max - 1;
      que[front] = x;
      num++;
```

```
        return x;
    }

    // enqueRear()  : 데이터를 rear에 enque
    public int enqueRear(int x) throws OverflowDeckException
    {
        if (num >= max)
            throw new OverflowDeckException();
        que[rear++] = x;
        if (rear == max)
            rear = 0;
        num++;
        return x;
    }

    // dequeFront() : front에서 deque
    public int dequeFront() throws EmptyDeckException
    {
        if (num <= 0)
            throw new EmptyDeckException();
        int x = que[front++];
        if (front == max)
            front = 0;
        num--;
        return x;
    }

    // dequeRear()  : rear에서 deque
    public int dequeRear() throws EmptyDeckException
    {
        if (num <= 0)
            throw new EmptyDeckException();
        if (--rear < 0)
            rear = max - 1;
        num--;
        return que[rear];
    }

    // peekFront()  : front 데이터를 peek
    public int peekFront() throws EmptyDeckException
    {
        if (num <= 0)
            throw new EmptyDeckException();
        return que[front];
    }

    // peekRear()   : rear 데이터를 peek
    public int peekRear() throws EmptyDeckException
    {
        if (num <= 0)
            throw new EmptyDeckException();
        return que[rear == 0 ? max - 1 : rear - 1];
    }

    // indexOf()
    public int indexOf(int x)
    {
        for (int i = 0; i < num; i++)
```

```
            if (que[(i + front) % max] == x)
                return i + front;
        return -1;
    }

    // 아래 메서드는 RingBufferQueue 코드와 동일
    // search()
    // clear()
    // capacity()
    // size()
    // isEmpty()
    // isFull()
    // dump()
}
```