<12.30 복습>

- QnA02.

```
BigInteger[] arr;
arr = new BigInteger[end];

for (int i = START; i < end; i++){
    arr[i] = BigInteger.valueOf((int)(Math.pow(2,i)));

    System.out.println("arr[" +i+ "] = "+arr[i]);
}
```

BigInteger.valueOf( )를 통해 어떤 숫자라도 BigInteger 형태로 변환이 가능하다.

!) arr이 BigInteger 배열인데 출력되는 값의 범위가
int값 표현 범위인 2147483648-1 = 2147483647까지라 의문이 생겼음.

해결-> (int)Math.pow(2,i)-> 이 값 자체가 int형으로 변환되었기 때문에
BigInteger로 변환하더라도 arr[31]부터 이후의 출력까지
int형 표현 범위인 2147483647만 나온다.

- QnA04.

1byte는 8bit로 2^8(256)개의 숫자를 표현할 수 있다.
컴퓨터는 0이라는 숫자를 표현하기 때문에 값의 범위에 0이 포함된다.
-> 8bit가 표현할 수 있는 수의 범위는 0~255까지이며,
8bit로 표현할 수 있는 최대값은 2^8 - 1 인 255이다.

- **10진수** : 0,1,2,3,4,5,6,7,8,9,10...17,18,19...
ex) 53 -> 10^1 x 5 + 10^0 x 3

- **16진수** : 0,1,2,3,4,5,6,7,8,9,10, ...15,10,11,12,13, ...18,19,110,111,112,113,114,115, ...
!) 11 1인지 1 11인지 1 1 1인지 구별이 안되는 상황이 발생하기 때문에
알파벳을 도입하여 [10 = a, 11 =b, 12 = c, 13 =d, 14 = e, 15 = f]로 표기한다.
-> 0,1,2,3,4,...,,8,9,a,b,c,d,e,f,10,11,12,13,14,...18,19,1a,1b,1c,1d,1e,1f, ...
+) 문자표기와 혼동되지 않도록 맨 앞에 0x를 붙여준다
ex) 255 -> 16^1 x 15 + 16^0 x 15 -> ff -> 0xff

-코드 분석

1. byte형 변수 limit 선언, byte형은 1byte = 8bit의 크기를 갖는다.

```
byte limit = (byte) 0xff;
System.out.printf("byte limit: 0x%x\n", limit);
System.out.format("0x%x%n", limit);
```

: 0xff -> 15 15 (16) == ff (16) -> 1111 1111 (2) = limit

지시자	설명
%b	불리언
%d	10진수
%o	8진수
%x, %X	16진수
%f	실수형 10진수
%e, %E	지수형태표현
%c	문자
%s	문자열
%n	개행

2. for문을 통해 1111 1111 출력해보기

```
for (int i = 0; i < 8; i++) {
    if (i != 0 && i % 4 == 0) {
        System.out.print(" ");
    }

    System.out.print(((0x80 >>> i) & limit) == 0 ? '0' : '1');
}
```

8비트이므로 -> 반복 횟수 8번
16진수는 2진수가 4개씩 묶인 것 -> if문을 사용해 I가 4의 배수일 때 출력구문 한 칸 띄우기

0x80 -> 8 0 (16) -> 1000 0000(2)
0x80 >>> I : I만큼 오른쪽 이동

(0x80>>> I) (limit)
I = 0 : 1000 0000 & 1111 1111 = 1000 0000 == 0? false -> 1 출력
I = 1 : 0100 0000 & 1111 1111 = 0100 0000 == 0? false -> 1 출력
I = 2 : 0010 0000 & 1111 1111 = 0100 0000 == 0? false -> 1 출력
I = 3 : 0001 0000 & 1111 1111 = 0100 0000 == 0? false -> 1 출력

I = 4 : 0000 1000 & 1111 1111 = 0100 0000 == 0? false -> 1 출력
I = 5 : 0000 0100 & 1111 1111 = 0100 0000 == 0? false -> 1 출력
I = 6 : 0000 0010 & 1111 1111 = 0100 0000 == 0? false -> 1 출력
I = 7 : 0000 0001 & 1111 1111 = 0100 0000 == 0? false -> 1 출력
I = 8 -> 루프 종료

출력

```
1111 1111
```

-비트 이동 연산자

x << y	정수 x의 각 비트를 y만큼 왼쪽으로 이동시킵니다. (빈자리는 0으로 채워집니다.)
x >> y	정수 x의 각 비트를 y만큼 오른쪽으로 이동시킵니다. (빈자리는 정수 a의 최상위 부호비트와 같은 값으로 채워집니다.
x >>> y	정수 x의 각 비트를 y만큼 오른쪽으로 이동시킵니다. (빈자리는 0으로 채워집니다.)

3. short형 변수 test 선언, short형은 2byte = 16bit의 크기를 갖는다.

```
short test = (short) 0x0fff;

System.out.printf("short limit: 0x%x\n", test);
System.out.format("0x%x%n", test);
```

: 0xfff -> 0 15 15 15(16) == 0fff (16) -> 0000 1111 1111 1111(2) = test

4. for문을 통해 1111 1111 출력

16비트이므로 -> 반복 횟수 16번
16진수는 2진수가 4개씩 묶인 것 -> if문을 사용해 I가 4의 배수일 때 출력구문 한 칸 띄우기

```
for (int i = 0; i < 16; i++) {
    if (i != 0 && i % 4 == 0) {
        System.out.print(" ");
    }

    System.out.print(((0x8000 >>> i) & test) == 0 ? '0' : '1');
}
```

0x8000 = 1000 0000 0000 0000
2번 과정과 동일, 반복횟수만 다름

5. 12비트로 표현할 수 있는 최댓값 확인

```
System.out.println("short test 2^12(4096) - 1: " + test);
```

0000 1111 1111 1111 (2)
-> 2^12 - 1 = 4096 - 1 = 4095