<문제은행 [3] 복습>

- **문제은행 [3]**

문제1.

Math.pow( ) 는 n승을 계산한다.
Math.pow(x,y) = x^y
ex) Math.pow(2,i) 는 2의 i승을 의미한다.

!) 배열의 31번 항까지 올바르게 계산된다.

- 나와야 하는 숫자
-2^30 = 1,073,741,824
-2^31 = 2,147,483,648
-2^32 = 4,294,967,296

- 실제 출력되는 숫자

```
수열의 몇 번째 항을 출력할까요? : 30
수열의 30번째 항 = 536870912
수열의 몇 번째 항을 출력할까요? : 31
수열의 31번째 항 = 1073741824
수열의 몇 번째 항을 출력할까요? : 32
수열의 32번째 항 = 2147483647
수열의 몇 번째 항을 출력할까요? : 33
수열의 33번째 항 = 2147483647
```

: 2^8 = 1byte = 256개
[표현할 수 있는 수의 범위 : -128 ~ -1 / 0 ~ 127]
0을 양수 쪽에 포함하고 있기 때문에 2^31의 끝자리가 홀수로 되어있다.
-> 결국 2^31 = 2,147,483,648를 표현하지 못하고 2^31 -1이 최대값이 된다.

- 2^30 = 1,073,741,824까지만 정상값이 출력된다.

```
seq[27] = 134217728
seq[28] = 268435456
seq[29] = 536870912
seq[30] = 1073741824
```

문제2.

문제1에서 표현할 수 있는 값의 범위를 초과해 비정상적인 값이 출력되는 문제는
무한에 가까운 값을 표현할 수 있는 BigInteger를 통해 해결할 수 있다.

```
final int START_IDX = 0;
final BigInteger BASE = new BigInteger( val: "2");

System.out.print("찾고자하는 수열의 항을 입력해주세요: ");

Scanner scan = new Scanner(System.in);
int idx = scan.nextInt();

BigInteger[] seq = new BigInteger[idx];
seq[START_IDX] = new BigInteger( val: "1");

for (int i = START_IDX + 1; i < idx; i++) {
    seq[i] = seq[i - 1].multiply(BASE);
    System.out.println("seq[" + i + "] = " + seq[i]);
}
```

```
seq[30] = 1073741824
seq[31] = 2147483648
seq[32] = 4294967296
seq[33] = 8589934592
seq[34] = 17179869184
seq[35] = 34359738368
seq[36] = 68719476736
seq[37] = 137438953472
seq[38] = 274877906944
seq[39] = 549755813888
```

문제3.

- 내가 푼 답과 다른 점: 중복 검사 유무

1. 100의 크기를 가지는 boolean형 배열 lottoBox를 만든다.

2. for문을 통해 당첨 자리 5개를 랜덤하게 할당한다. (100명 중 5명 당첨)
   할당된 숫자는 selectIdx 배열에 들어간다.
   그리고 lottoBox의 인덱스가 되어 해당 위치의 값을 True로 바꾼다

```
final int TOTAL = 100;
final int SELECT = 5;

boolean[] lottoBox = new boolean[TOTAL];
int[] selectIdx = new int[SELECT];

lottoIdx = (int) (Math.random() * TOTAL);
```

3. 할당된 자리 중 중복이 존재할 가능성 존재하기 때문에 중복에 대한 검사 필요
   선택된 인덱스는 0~99 사이의 랜덤값이다.
   어떻게 이 랜덤 인덱스의 중복 여부를 판정할 것인가?
   -> 실제 SELECT는 5개, SELECT를 활용한 5개 배열에 할당된 랜덤 인덱스를 배치
   최악의 경우로 가정하더라도 검사는 최대 4회만 수행되면 된다.
: 현재 발생된 난수값이 5개 배열에 존재하는지 검사!

- 중복이 발생했는지 판단하는 boolean 변수 : isRealloc
  난수 발생 후 - 중복 발생 -> isRealloc = true
  - X -> isRealloc = false
  -> for문을 통해 현재 발생된 난수값과 이전에 발생한 난수값들이 들어있는 selectIdx 배열의 값을 비교해 중복을 검사한다.

+ allocCnt라는 변수는 난수값이 실제로 할당되었을 때 증가되는 값으로
현재 난수값이 몇 개 존재하는지 확인할 수 있으며, 새로운 난수값이 할당되었을 때
중복검사를 몇 번 수행할지 결정한다.
// 처음에 중복이 판정되면 나머지까지 검사할 필요가 없다.

+ 배열 선언이나 반복조건에 선언해둔 END 변수를 활용해야 한다.

```
final int END = 5;

int randIdxArr[] = new int[5];
  while(i < 5){
for(i=0;i<5;i++){
```

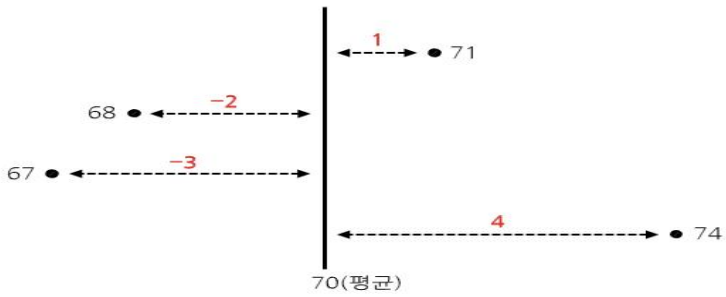
문제4.

주의할 점-> 최종 출력 시 sum을 float 처리해야한다.

```
System.out.println("반 평균 = " + (float)(sum) / STUDENT_NUM);
```

문제 5.

모분산 : 코드작성자가 운영자일 때 (데이터 가지고 있을 때) 적합
표본분산: 코드작성자가 샘플만 가지고 분석을 할 때 적합



$$\frac{1^2 - 2^2 - 3^2 + 4^2}{4} = 7.5$$

분산: 각 편차들(값과 평균치의 차이)들을 제곱한 다음 더해주고 전체 숫자로 나눈 값
표준 편차 : 분산에 루트를 씌워준 값

- Math 클래스

: Math 클래스는 수학에서 자주 사용하는 상수들과 함수들을 미리 구현해놓은 클래스
Math클래스의 모든 메소드는 클래스 메소드(static method)이므로, 객체를 생성하지 않고도
바로 사용할 수 있다.

1. random( ) 메소드 : 0.0 이상 1.0 미만의 범위에서 임의의 double형 값을 하나 생성하여 반환
   ex) Math.random( );

2. abs( )메소드 : 전달된 값의 절댓값을 반환한다.
   ex) Math.abs(-10); -> 10

3. floor( ), ceil( ), roun( )
- floor메소드는 인수로 전달받은 값과 같거나 작은 수 중에서 가장 큰 수를 반환
ex) Math.floor(10.9); -> 10.0
- ceil메소드는 반대로 인수로 전달받은 값과 같거나 큰 수 중에서 가장 작은 정수를 반환
ex) Math.ceil(10.1); -> 11.0
- round메소드는 전달받은 실수를 소수점 첫째 자리에서 반올림한 정수를 반환
ex) Math.round(10.5); -> 11

4. max( )와 min( )
- max메소드는 전달된 두 값을 비교하여 그 중에서 큰 값을 반환
ex) Math.max(3.4); -> 4
- min메소드는 그 중에서 작은 값을 반환
ex) Math.max(3.4); -> 3

5. pow()와 sqrt()
- pow()는 전달된 두 개의 double형 값을 가지고 제곱 연산을 수행
ex) (int)Math.pow(5,2) -> 25
- sqrt()는 전달된 double형 값의 제곱근 값을 반환
ex) (int)Math.sqrt(25) -> 5

6. sin( ), cos( ), tan( )
전달된 double형 값의 사인값, 코사인값, 탄젠트값 반환

문제6.

- 내가 푼 답과 다른 점: 길이를 구해 배열의 크기를 정함

-코드 분석

1. 배열에 저장할 숫자 지정

```
BigInteger testNum = new BigInteger( val: "45678911234");
```

!) BigInteger는 문자열로 되어 있기 때문에 초기화를 위해 문자열을 인자값으로 넘겨주어야 한다.

2. testNum값을 저장할 배열의 길이 구하기
ex) 100을 배열에 넣으려고 할 때 배열의 크기 구하기
log_10(100) = 2 + 1 -> 길이 3을 구할 수 있다.

- 계산에 사용할 변수

```
final BigInteger BASE = TEN;
BigInteger mantissa = ZERO;
// 초기 testNum을 10으로 나눠서 몫이 있는지 검사함
BigInteger n = testNum.divide(TEN);
```

몫을 구하기 위한 BASE = 10
배열의 길이를 구하기 위한 mantissa값 = ZERO
testNum을 10으로 나눈 몫이 저장되는 변수 n

- 길이 계산

```
while (n.compareTo(ZERO) == 1) { // n이 0 보다 크면 여전히 10^n 으로 나눌 수 있으므로 계속 나눔
    n = n.divide(TEN);          // 두 번째 10 으로 나누기 (즉 100 나누기), 다음 1000 나누기 ...
    mantissa = mantissa.add(ONE); // log10(100) = 2 이므로
}
```

길이 계산 반복횟수에 대한 조건 필요하다.
!) 두 수를 비교하는 compareTo를 활용한다.
n.compareTo(ZERO)
: n이 0보다 더 크면 1, 같으면 0을 반환한다.

초기 n값인 4567891123를 0과 비교했을 때 n이 더 크므로 결과는 1이 된다. -> 최초 계산 수행
n이 0보다 클 때 반복계산을 시키기 위해 while문의 동작조건을 (n.compareTo(ZERO) == 1)로 지정
while 내부 동작 = n을 10으로 몫을 다시 n에 대입하고, mantissa 값을 1 증가시킨다.

n의 값이 0이 될 때까지 루프가 진행된다.
//마지막 10^0 자리에 해당하는 4를 10으로 나눴을 때 취해지는 몫은 0이 되고,
  ZERO와 같아져 반복이 종료된다.

while 내부 동작

```
4567891123
456789112
45678911
4567891
456789
45678
4567
456
45
4
```

mantissa 값을 int형으로 변환해 int형 변수 length에 대입한다.

```
int length = mantissa.intValue();   // 최종 결과에 + 1을 해야함 (초기 10 나눈건 계산 안함)
System.out.println("456789112345의 길이: " + (length + 1));

int[] numArr = new int[length + 1];
```

3. testNum의 숫자를 배열에 대입한다.
- testNum에 들어간 숫자의 각 자리수에 맞는 숫자를 배열에 배치시켜야 한다.

ex) 1234 = 1 x 10^3 + 2 x 10^2 + 3 x 10^1 + 4 x 10^0
-> arr[0] = 4, arr[1] = 3, arr[2] = 2, arr[3] = 1
1234 / 10^1 = 123 -> 123 % 10 = 3 -> arr[1] = 3

-> 10^n으로 나눠 몫을 구한 뒤, 구해진 몫에 나머지 연산을 수행해 저장할 값을 지정한다.

for문을 사용해 위에서 구한 길이만큼 계산과 대입이 반복된다.

```
for (int i = length; i >= 0; i--) {
```

```
numArr[i] = testNum.divide(
        new BigInteger(
                String.valueOf(
                        //Math.pow(BASE, i)
                        BASE.pow(i)
                )
        )
).mod(
        TEN
).intValue();   // BigInteger를 int로 변환함
```

testNum을 BASE^i로 나눠 몫을 구한 뒤, 여기에 나머지 연산을 수행한다.
배열은 int형이므로 최종값에 .intValue( )를 수행해 연산의 결과를
int형으로 바꿔 numArr[i]에 저장한다.

!) String.valueOf() - 오브젝트의 값을 String으로 변환한다.
Object가 NULL인 경우
Null PointerException(NPE)을 발생시키는 toString()과 달리 valueOf는 null이라는 문자열로 처리한다.

- BigInteger 연산

0	ZERO		BigInteger.ZERO
1	ONE		BigInteger.ONE
10	TEN		BigInteger.TEN
		연 산	
a + b	더하기		a.add(b)
a - b	빼기		a.subtract(b)
a * b	곱하기		a.multiply(b)
a / b	나누기		a.divide(b)
a % b	나머지		a.mod(b)
a % b	나머지 (음수)		a.remainder(b)

-BigInteger 형 변환

1. int

ex) bigNumber.intValue();

2. long

ex) bigNumber.longValue();

3. float

ex) bigNumber.floatValue();

4. double

ex) bigNumber.doubleValue();

5. String

ex) bigNumber.toStringValue();

-BigInteger 두 수 비교

compareTo

ex) int compare = bigNumber1.compareTo(bigNumber2);

참고: http://egloos.zum.com/js7309/v/11114056

문제8.

- 내가 푼 답과 다른 점

1. 게임 진행 방식
- 강사님 코드 : 한 판으로 승자 결정

```
상대방 주사위 눈금을 2 떨군다.
dice[0] = 9
dice[1] = 6
플레이어 1 승리!

Process finished with exit code 0
```

- 내 코드 : 3판 2선승

```
-----------------------------------------------------
!) 1번째 판이 시작됩니다.
-----------------------------------------------------
player1가 주사위를 굴립니다.
첫 번째 주사위의 값 = 4
!) 특수주사위를 굴립니다.
-> 특수주사위의 눈금값 = 2
★ 특수스킬2가 발동되어 특수주사위값2가 총합에 더해집니다. ★
두 번째 주사위의 값 = 3
던진 주사위의 총합 = 9
-----------------------------------------------------
-----------------------------------------------------
player2가 주사위를 굴립니다.
첫 번째 주사위의 값 = 3
두 번째 주사위의 값 = 5
던진 주사위의 총합 = 8
-----------------------------------------------------
=====================================================
abc의 주사위합 = 9, ewq의 주사위합 = 8
=====================================================
1라운드의 승리자 : abc
현재 abc의 승점 = 1
!) 1번째 판이 종료되었습니다.
=====================================================
```

```
-----------------------------------------------------
 - 승부 기록 -
 1번째 판 승리자: abc / 2번째 판 승리자: abc / 3번째 판 승리자: null /
 -----------------------------------------
 -> 최종 승리자는 abc입니다!
 Process finished with exit code 0
```

2. 특수주사위 발동 조건 : 내 코드는 각 판에서 첫 번째 주사위 값이 짝수일 때만 발동
- 강사님 코드

```
플레이어1의 차례
플레이어1의 1번째 주사위값 = 4
플레이어1의 주사위합 = 4
플레이어1의 2번째 주사위값 = 2
플레이어1의 주사위합 = 6
플레이어2의 차례
플레이어2의 1번째 주사위값 = 4
플레이어2의 주사위합 = 4
플레이어2의 2번째 주사위값 = 6
플레이어2의 주사위합 = 10
플레이어 1의 주사위합은 6입니다. 특수주사위를 굴립니다.
```

- 내 코드

```
-------------------------------------------------------
!) 1번째 판이 시작됩니다.
-------------------------------------------------------
player1가 주사위를 굴립니다.
첫 번째 주사위의 값 = 2
!) 특수주사위를 굴립니다.
-> 특수주사위의 눈금값 = 1
★ 특수스킬1이 발동되어 poi의 눈금값을 2 떨굽니다.. ★
두 번째 주사위의 값 = 5
던진 주사위의 총합 = 7
-------------------------------------------------------
-------------------------------------------------------
player2가 주사위를 굴립니다.
첫 번째 주사위의 값 = 2
!) 특수주사위를 굴립니다.
-> 특수주사위의 눈금값 = 4
★ 특수스킬4가 발동되어 poi가 패배합니다.. ★
=======================================================
abc의 주사위합 = 11, poi의 주사위합 = 10
=======================================================
1라운드의 승리자 : abc
현재 abc의 승점 = 1
!) 1번째 판이 종료되었습니다.
=======================================================
```

3. 음수 처리 유무

- **코드 분석**

1. 변수 선언

사용자 수를 의미하는 PLAYER_NUM
굴릴 주사위의 개수를 의미하는 DICE_NUM = 2

```
final int PLAYER_NUM = 2;
final int DICE_NUM = 2;
```

특수 스킬 4개에 해당하는 눈금값이 지정되어 있는 SKILL_NUM1 ~ SKILL_NUM4
특수스킬 4가 나올 경우 특수주사위를 굴린 사용자를 패배시키기 위한 변수 DEATH

```
final int SKILL_NUM1 = 1;
final int SKILL_NUM2 = 3;
final int SKILL_NUM3 = 4;
final int SKILL_NUM4 = 6;

final int DEATH = 4444;
```

주사위 눈금값에 대한 범위 지정

```
final int MAX = 6;
final int MIN = 1;
int range = MAX - MIN + 1;
```

주사위와 주사위 눈금합을 저장할 배열 선언

```
int dice;
int[] diceSum = new int[PLAYER_NUM];
```

2. 주사위 굴리기

이중 for문을 사용
첫 번째 for문은 사용자 수만큼 반복시키기 위함
두 번째 for문은 주사위 굴리기 횟수만큼 반복시키기 위함
두 번째 for문 내부에서 주사위값을 난수로 Math.random()을 사용해 발생시키고
주사위 눈금합을 저장하는 배열인 diceSum에 순차적으로 저장

```
for (int i = 0; i < PLAYER_NUM; i++) {
    System.out.printf("플레이어%d의 차례\n",i+1);
    // 많이들 실수하는 부분임
    // for문 내부에서 i랑 j랑 바꿔 쓰는 경우를 조심합시다!!!
    for (int j = 0; j < DICE_NUM; j++) {
        dice = (int) (Math.random() * range + MIN);
        System.out.printf("플레이어%d의 %d번째 주사위값 = %d\n",i+1,j+1,dice);
        diceSum[i] += dice;
        System.out.printf("플레이어%d의 주사위합 = %d\n",i+1,diceSum[i]);
    }
}
```

```
플레이어1의 차례
플레이어1의 1번째 주사위값 = 2
플레이어1의 주사위합 = 2
플레이어1의 2번째 주사위값 = 1
플레이어1의 주사위합 = 3
플레이어2의 차례
플레이어2의 1번째 주사위값 = 5
플레이어2의 주사위합 = 5
플레이어2의 2번째 주사위값 = 4
플레이어2의 주사위합 = 9
```

3. 주사위를 굴린 후 특수 주사위 발동 여부 판정

각 플레이어들의 주사위 합을 비교해 특수 주사위 발동 여부 판정
모든 플레이어들의 주사위값을 비교하기 위해 for문 사용, 지정된 사용자 수만큼 반복시킨다.
diceSum 배열에 저장된 값이 짝수라면 특수주사위를 굴린다.
짝수가 아니라면 현재 diceSum 배열에 저장된 값으로 승자를 결정한다.

```
for (int i = 0; i < PLAYER_NUM; i++) {

    if (diceSum[i] % 2 == 0) {
        System.out.printf("플레이어 %d의 주사위합은 %d입니다. 특수주사위를 굴립니다.\n",i+1,diceSum[i]);
        dice = (int) (Math.random() * range + MIN);
        System.out.printf("특수주사위 눈금값 = %d\n",dice);
```

```
else {
    diceSum[i] += dice;
}
```

4. 특수 스킬

- 스킬 1 : 특수주사위 값 = 1

```
if (dice == SKILL_NUM1) {
    System.out.println("skill1 : 상대방 주사위 눈금을 2 떨군다.");

    for (int j = 0; j < PLAYER_NUM; j++) {
        if (j == i) {
            continue;   // skip의 의미임
        }
        diceSum[j] -= 2;
    }
}
```

for문 사용 , j값은 플레이어의 수만큼 증가
현재 플레이어 i에 해당하지 않으면 플레이어[j]의 주사위합을 2 감소시킨다.
- ‘ **현재 플레이어에 해당하지 않으면** ‘에 대한 조건 필요
-> if(j == I)
i 값은 현재 플레이어를 의미, 현재 플레이어와 다음 플레이어 모두를 의미하는
증가하는 j값을 i와 비교한다.
: 루프 안에서 현재 j값이 i값과 같다면 현재 j값은 자기 자신을 의미하기 때문에 skip
-> continue를 사용
j와 I가 다르다면 상대 플레이어를 의미하므로 상대 플레이어의 주사위합을 2 감소 시킨다.

- 스킬2 : 특수 주사위 값 = 3

```
else if (dice == SKILL_NUM2) {
    System.out.println("skill3 : 모두 함께 자폭 ^^ -6");

    for (int j = 0; j < PLAYER_NUM; j++) {
        diceSum[j] -= 6;
    }
}
```

diceSum 배열에 저장된 모든 값을 6 감소시키기 위해 for문 사용

-스킬3 : 특수 주사위 값 = 4

```
else if (dice == SKILL_NUM3) {
    System.out.println("skill4 : 그냥 가세요 ㅠㅜ"); //패배 처리

    diceSum[i] = DEATH;
}
```

현재 플레이어를 의미하는 I의 주사위합을 DEATH 값으로 바꿔 패배시킨다.

-스킬4 : 특수 주사위 값 = 6

```
else if (dice == SKILL_NUM4) {
    System.out.println("skill6 : 모두에게서 3씩 뺏어서 내거에 추가한다.");

    for (int j = 0; j < PLAYER_NUM; j++) {
        if (i == j) {
            continue;
        }

        diceSum[j] -= 3;
        diceSum[i] += 3;
    }
}
```

스킬 1(특수주사위값=1)과 마찬가지로 for문 사용
현재 비교하는 값이 자기 자신인지 상대방인지 판단하는 조건 사용
- 자기 자신이면 skip하고 다음 j값으로 넘어가 다음 플레이어의 주사위합인 dice[j]에서 3을 빼고
  현재 플레이어의 주사위합인 dice[i]에 3을 더한다.

- 음수 처리

```
for (int i = 0; i < PLAYER_NUM; i++) {
    if (diceSum[i] < 0) {
        diceSum[i] = 0;
    }

    System.out.printf("dice[%d] = %d\n", i, diceSum[i]);
}
```

음수가 나오지 않도록 하기 위해 플레이어들의 주사위 합을 검사한다.
주사위 합이 0보다 작다면 0으로 바꾼다.

4. 승부 판정

- 특수스킬3이 발동되었는지 확인

```
boolean checkWinner = true;

for (int i = 0; i < PLAYER_NUM; i++) {
    if (diceSum[i] == DEATH) {
        System.out.printf("플레이어%d가 패배하였습니다!\n", i);
        checkWinner = false;
    }
}
```

승부를 판정하기 전 특수스킬3이 발동되었는지 확인이 필요하다.
for문을 사용해 플레이어들의 주사위합에 4444가 들어있는지 검사한다.
승부 판정은 checkWinner 조건이 true일 때만 수행되기 때문에 특수스킬3이 발동되었을 경우에만 checkWinner를 false로 만들어 아래의 승부판정이 수행되지 않도록 한다.

- 특수스킬 3이 발동되지 않아 승부판정이 필요할 때

```
if (checkWinner) {
    if (diceSum[0] > diceSum[1]) {
        System.out.println("플레이어 1 승리!");
    } else if (diceSum[0] < diceSum[1]) {
        System.out.println("플레이어 2 승리!");
    } else {
        System.out.println("무승부!");
    }
}
```

특수스킬 3이 발동되지 않았을 경우에 진행되는 승부판정 코드
diceSum 배열에 저장된 값들을 비교해 값이 가장 큰 사람을 승자로 지정한다.

-Q&A
< if vs for >
for: 반복적으로 검사가 필요할 때
if : 반복 검사가 필요 없을 때
ex) 문제8은 반복적인 검사가 필요해 for 내부에서 if를 사용하고 있다.