<문제은행 [3] _ 클래스화>

1. 아래와 같은 등비 수열이 있다.
   1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, ...
   사용자 입력을 통해 원하는 위치의 값을 뽑아내도록 프로그래밍 해보자!
2. 1번 문제에서 32번째 항이 21억 정도가 나올 것이다.
   BigInteger를 통해서 50번째 항을 구해보자!

[MyCustomSequenceTest(main) - MyCustomSequence - NonInitialSet]

-MyCustomSequenceTest(main)

```
public class MyCustomSequenceTest {
    public static void main(String[] args) {
        final int ORDER = 31;
        final int ORDER2 = 50;

        MyCustomSequence mcs = new MyCustomSequence( equalRatio: 2);

        // 2^0 <<<
        System.out.printf("2^%d = %d\n", ORDER - 1, mcs.getGeoSeqNthOrderData(ORDER));
        System.out.printf("2^%d = %d\n", ORDER2 - 1, mcs.getGeoSeqNthOrderData(ORDER2));
    }
}
```

1. 몇 번째 항을 구할 건지 정하는 ORDER 변수 2개

2. MycustomSequence 클래스의 의 생성자 호출해서 MycustomSequence타입의 객체 mcs 생성 (equalRatio 값(2)를 넘겨준다)

3. ORDER(31) 값을 넘겨주며 MycustomSequence의 getGeoSeqNthOrderData 메소드를 호출한다.

4. ORDER2(50) 값을 넘겨주며 MycustomSequence의 getGeoSeqNthOrderData 메소드를 호출한다.

- MyCustomSequence

```java
import java.math.BigInteger;

public class MyCustomSequence {
    InitialSet is;
    NonInitialSet nis;

    public MyCustomSequence (final int[] arr, int bias) { is = new InitialSet(arr, bias); }
    public MyCustomSequence (final int[] arr, int bias, int jumping) { is = new InitialSet(arr, bias, jumping); }

    public MyCustomSequence (final int equalRatio) { nis = new NonInitialSet(equalRatio); }

    public int getNthOrderData (int count) { return is.getNthOrderData(count); }

    public BigInteger getGeoSeqNthOrderData (int count) { return nis.getGeoSeqNthOrderData(count); }
}
```

1. NonInitialSet 타입의 객체 nis 선언

2. 생성자

```java
public MyCustomSequence (final int equalRatio) { nis = new NonInitialSet(equalRatio); }
```

main에서 equalRatio값을 받아온다
NonInitialSet 타입의 객체 nis의 생성자를 호출하며 받아온 equalRatio 값을 넘겨준다.

3. getGeoSeqNthOrderData

```java
public BigInteger getGeoSeqNthOrderData (int count) { return nis.getGeoSeqNthOrderData(count); }
```

main에서 전달받은 ORDER or ORDER2을 count로 설정하고
count를 넘겨주며 is객체의 getGeoSeqNthOrderData 메소드를 호출한다.
메소드의 리턴값은 BigInteger 값

- NonInitialSet (BigInteger)

```
public class NonInitialSet {
    int equalRatio;

    public NonInitialSet (final int equalRatio) { this.equalRatio = equalRatio; }
```

1. 변수 선언
 : 등비 equalRatio 선언

2. 생성자
 : MyCustomSequence에서 받아온 equalRatio값을 NonInitialSet 클래스의 변수 equalRatio에 대입한다.

3. getGeoSeqNthOrderData

```
public BigInteger getGeoSeqNthOrderData (final int count) {
    // 디버깅용(값이 올바르게 나오나 체크)
    for (int i = 0; i < count; i++) {
        System.out.printf("%d^%d = %d\n", equalRatio, i,
                BigInteger.valueOf(equalRatio).pow(i));
    }

    return BigInteger.valueOf(equalRatio).pow(count - 1);
}
```

: MyCustomSequence 클래스에서 count (31 or 50)값을 받아온다.

for문을 사용해 0부터 지정된 count값까지의 등비수열을 출력한다.
equalRatio(2)값을 BigInteger 형식으로 바꿔 계산한다.

반환값은 BigInteger 형식의 값 2^30 or 2^50

3. 배열로 로또 문제를 만들어보기!
   실제 로또 확률은 0.00000023으로 1억명중 23명이 당첨됨
   실제값을 사용하기엔 검토 작업이 너무 고통스러우므로 100명 중 5명을 뽑아보도록 하자!
   배열값에 당첨되는 자리를 배치해놓고 사용자가 돌려서 당첨되는지 안되는지를 판정하도록 한다.
9. 문제 은행 [2]의 3번을 배열화하여 풀어보자!

-> 랜덤 경매장
1. 들어오는 사람들에게 랜덤으로 번호를 할당해준다.
2. 추첨 기계가 랜덤으로 번호를 할당한다.
3. 당첨된 사람들을 매칭시킨다.

[Bank3Ans3ClassTest (main) - MyLotto]
[Bank3Ans3ClassTest (main) - MyLottoRefactor]

- Bank3Ans3ClassTest (main)

```
MyLotto myLotto = new MyLotto( totalNum: 100,  selectNum: 5);

myLotto.allocPeopleNumber();
//myLotto.printPeopleArr();

myLotto.raffle();
System.out.println("=================== 경계선 ===================");

MyLottoRefactor mlr = new MyLottoRefactor( totalNum: 100,  selectNum: 5);

mlr.raffle2();
```

totalNum과 selectNum을 넘겨주며 MyLotto 형식의 객체인 myLotto를 생성한다.
myLotto 객체의 raffle 메소드를 호출한다.

totalNum과 selectNum을 넘겨주며 MyLottoRefactor 형식의 객체인 mlr을 생성한다.
mlr 객체의 raffle2 메소드를 호출한다.

- MyLotto

1. 변수 선언

```
public class MyLotto {
    int totalNum;
    int selectNum;

    int[] peopleArr;
    int[] selectArr;

    int max, range;
    final int MIN = 1;
```

전체 숫자의 갯수를 의미하는 totalNum
당첨 숫자의 갯수를 의미하는 selectNum
전체 인원을 지정할 배열 peopleArr
당첨된 숫자를 저장할 배열 selectArr
난수의 범위와 관련된 변수 max, range, MIN

2. 생성자

```
public MyLotto (final int totalNum, final int selectNum) {
    this.totalNum = totalNum;
    this.selectNum = selectNum;

    max = totalNum;
    range = max - MIN + 1;

    peopleArr = new int[totalNum];
    selectArr = new int[selectNum];
}
```

main으로부터 totalNum과 selectNum을 받아와 MyLotto 클래스의
totalNum과 selectNum에 대입한다.
max값과 range값을 설정한다.
totalNum만큼의 크기를 가지는 peopleArr 배열을 생성한다.
selectNum만큼의 크기를 가지는 selectArr 배열을 생성한다.

3. allocPeopleNumber

```
public void allocPeopleNumber () {
    int rand;

    for (int i = 0; i < totalNum; i++) {
        do {
            rand = (int) (Math.random() * range + MIN);
        } while (isDuplicate(rand, peopleArr, i));

        peopleArr[i] = rand;
    }
}
```

1~100까지의 난수를 저장할 배열 rand를 선언한다.
클래스의 totalNum변수는 전체 숫자를 의미하며,
생성자가 호출되며 이 값은 100으로 지정되었다.
100개의 난수값을 만들어야 하기 때문에 for문을 통해 난수를 100번 발생시키고
각 루프에서 발생된 난수를 peopleArr 배열에 저장한다.

중복된 숫자가 당첨되지 않도록 난수에 대한 검사가 필요하다.
do_while문을 사용해 발생된 난수에 대한 중복검사를 수행한다.
while의 조건문에는 isDuplicate 메소드를 넣어주어 현재 난수값이 중복되지 않았다는
의미인 false를 리턴 받을 때까지 [난수 발생 - 중복 검사]의 루틴이 반복된다.
+ isDuplicate에 전달해주는 변수는 rand, peopleArr, 현재 I 값이다.

```
 23  81  58  16  64  54  90  34  62  43
 31  86  28  39  26  53  82  41  75  80
 70  98  52  78  24   2  76  71  44  67
 68  65  77  30  25   6  69  89   8  14
 66  37  21  97  72  42  91  92  49  35
  4 100  38  59  18  96  27  99  32  47
 10  45  40   5  63  33  88  15  73  56
 17   9  83  93  84  57  94  79  29  22
 87   1  13  50  55  74  46  11  36  95
 61   3  60  85  51  48  19   7  20  12
```

- isDuplicate

```
public boolean isDuplicate (int rand, int[] arr, int cnt) {
    for (int i = 0; i < cnt; i++) {
        if (arr[i] == rand) {
            return true;
        }
    }

    return false;
}
```

전달받은 rand, peopleArr배열, 현재 I 값을 검사에 필요한 임시배열과 변수로 지정한다.
arr배열은 peopleArr 배열의 정보를 가지고,
cnt는 진행된 반복만큼의 I값을 가진다.
ex) rand = 10, peopleArr = { 10, 11 } , I = 2
arr = { 10, 11 } , cnt = 2
arr의 0번 인덱스부터 1번 인덱스까지 현재 난수값인 rand = 10과 일치하는 값이
있는지 확인한다. 같은 값이 있다면 false를 리턴하고, 없다면 true를 리턴한다.

4. raffle

```
public void raffle () {
    allocSelectNumber();
    checkWinner();
}
```

MyLotto 클래스의 raffle 변수는
allocSelectNumber 메소드와 checkWinner 메소드를 순차적으로 호출한다.

- allocSelectNumber

```
public void allocSelectNumber () {
    int rand;

    for (int i = 0; i < selectNum; i++) {
        do {
            rand = (int) (Math.random() * range + MIN);
        } while (isDuplicate(rand, selectArr, i));

        selectArr[i] = rand;
    }
}
```

난수값을 저장할 배열 rand를 선언한다.
클래스의 selectNum변수는 당첨 숫자의 개수를 의미하며,
생성자가 호출되며 이 값은 5로 지정되었다.
5개의 당첨숫자를 만들어야 하기 때문에 for문을 통해 난수를 5번 발생시키고
각 루프에서 발생된 난수를 당첨 숫자가 저장되는 배열인 selectArr에 저장한다.

중복된 숫자가 당첨되지 않도록 당첨 숫자에 대한 검사가 필요하다.
do_while문을 사용해 발생된 난수에 대한 중복검사를 수행한다.
while의 조건문에는 isDuplicate 메소드를 넣어주어 현재 난수값이 중복되지 않았다는
의미인 false를 리턴 받을 때까지 [난수 발생 - 중복 검사]의 루틴이 반복된다.
+ isDuplicate에 전달해주는 변수는 rand, selctArr, 현재 I 값이다.

- isDuplicate

```java
public boolean isDuplicate (int rand, int[] arr, int cnt) {
    for (int i = 0; i < cnt; i++) {
        if (arr[i] == rand) {
            return true;
        }
    }

    return false;
}
```

전달받은 rand, selctArr배열, 현재 I 값을 검사에 필요한 임시배열과 변수로 지정한다.
arr배열은 당첨된 숫자들이 저장된 selectArr 배열의 정보를 가지고,
cnt는 진행된 반복만큼의 I값을 가진다.

ex) rand = 10, selectArr = { 10, 11 } , I = 2
arr = { 10, 11 } , cnt = 2

arr의 0번 인덱스부터 1번 인덱스까지 현재 난수값인 rand = 10과 일치하는 값이
있는지 확인한다. 같은 값이 있다면 false를 리턴하고, 없다면 true를 리턴한다.

5. checkWinner

```java
public void checkWinner () {
    for (int i = 0; i < selectNum; i++) {
        for (int j = 0; j < totalNum; j++) {
            if (selectArr[i] == peopleArr[j]) {
                System.out.printf("위치 %d 당첨!!!\n", j);
                break;
            }
        }
    }
}
```

이중 for문을 통해 현재 selectArr 배열의 인덱스 I의 값과
peopleArr 배열의 인덱스 j의 값이 일치하는지 비교해
일치하는 경우 현재 위치 j가 당첨이라는 메시지를 출력하고 j 반복문을 빠져나가
다음 당첨숫자로 넘어간다.
+ 인덱스 j는 위치를 의미한다.
인덱스 j의 값이 selectArr의 인덱스 I의 값과 같은지 비교해
같으면 인덱스 j 위치 당첨

```
  9  27  33   1  14  54  18  90  68  20
 88   5  73  40  31  35  47  56  13  39
 26  43  51   3  71  69 100  84  70  19
 85   8  16  87  76  17  49  66  30  36
 10  91  60  55  97  41  25  95  82  12
 37  63  89  74  53  86  23  45  59  96
 65  34  67  62  52  75  28  93  32  94
 38  42  50  11  58  98  83  57  92  48
 44  79  22  72  61  24   7  21   6  64
  4  77  46  29   2  78  81  99  80  15
당첨 =46
당첨 =27
당첨 =49
당첨 =93
당첨 =36
위치 92 당첨!!!
위치 1 당첨!!!
위치 36 당첨!!!
위치 67 당첨!!!
위치 39 당첨!!!
```

- MyLottoRefactor

1. 변수 선언

```
int totalNum;
int selectNum;

int[] peopleArr;
int[] selectArr;

int max, range;
final int MIN = 1;
```

2. 생성자

```
public MyLottoRefactor(final int totalNum, final int selectNum) {
    this.totalNum = totalNum;
    this.selectNum = selectNum;

    max = totalNum;
    range = max - MIN + 1;

    peopleArr = new int[totalNum];
    selectArr = new int[selectNum];
}
```

3. raffle2

```
public void raffle2 () {
    allocNonDuplicateArrNumber(totalNum, peopleArr);
    allocNonDuplicateArrNumber(selectNum, selectArr);
    checkWinner();
}
```

!) MyLotto 클래스와 다르게
하나의 메소드로 매개변수만 바꿔서 난수배열을 처리한다.

- allocNonDuplicateArrNumber

```
public void allocNonDuplicateArrNumber (final int cnt, int[] arr) {
    int rand;

    for (int i = 0; i < cnt; i++) {
        do {
            rand = (int) (Math.random() * range + MIN);
        } while (isDuplicate(rand, arr, i));

        arr[i] = rand;
    }
}
```

- isDuplicate

```
public boolean isDuplicate (int rand, int[] arr, int cnt) {
    for (int i = 0; i < cnt; i++) {
        if (arr[i] == rand) {
            return true;
        }
    }

    return false;
}
```

- checkWinner

```
public void checkWinner () {
    for (int i = 0; i < selectNum; i++) {
        for (int j = 0; j < totalNum; j++) {
            if (selectArr[i] == peopleArr[j]) {
                System.out.printf("위치 %d 당첨!!!\n", j);
                break;
            }
        }
    }
}
```

6. 45678911234라는 숫자를 BigInteger에 배치한다.
   각 자리수에 맞는 숫자를 배열에 배치하도록 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

[Bank3Ans6ClassTest(main) , BigMath]

- Bank3Ans6ClassTest(main)

```
import java.math.BigInteger;

public class Bank3Ans6ClassTest {
    public static void main(String[] args) {
        int num = BigMath.log10(new BigInteger( val: "45678911234"));
        System.out.println("이야 되네 ? " + num);

        BigMath.bigLocationCheck();
    }
}
```

유틸리티 클래스인 BigMath의 메소드 log10의 결과값(배열의 길이)을 num에 대입한다.
BigMath 클래스의 bigLocationCheck 메소드를 호출한다.

- BigMath

!) 유틸리티 클래스

인스턴스 메서드와 인스턴스 변수를 제공하지 않고 정적 메서드와 변수만을 제공하는 클래스.
공통적으로 사용되는 메서드를 모아서 클래스로 만든다.
(데이터와 데이터 처리를 위한 로직의 캡슐화 (x)
 비슷한 기능의 메서드와 상수를 모아서 캡슐화(o))

-> 중복된 코드 없이 효율적으로 관리 가능
유틸리티 클래스의 모든 메서드들은 static이며 이 클래스는 인스턴스화될 필요가 없으므로
이를 방지하기 위해 기본 생성자를 private로 만들며, 상속을 방지하기 위해 클래스를 final로 선언한다.

1. 변수 선언

```
static int arrLength;
static BigInteger bigNum;
```

배열의 길이를 의미하는 arrLength
배열에 저장할 숫자를 의미하는 bignum

- log10

```
static public int log10 (BigInteger num) {
    bigNum = num;
    BigInteger mantissa = ZERO;
    BigInteger n = num.divide(TEN);

    int tmp;

    while (n.compareTo(ZERO) == 1) {
        n = n.divide(TEN);
        mantissa = mantissa.add(ONE);
    }

    tmp = mantissa.intValue();
    arrLength = tmp + 1;

    return tmp;
}
```

main에서 bigInteger 값인 45678911234를 받아와 bignum에 대입한다.
배열의 길이를 구하기 위한 mantissa 값 = ZERO
num을 10으로 나눈 몫이 저장되는 변수 n
길이계산값을 의미하는 tmp

- 길이 계산
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.

mantissa 값을 int형으로 변환해 int형 변수 tmp에 대입한다.
!) arrLength = tmp + 1 : 초기 10을 나눈 것까지 길이에 반영해야 한다.

tmp 값을 반환한다.

```
이야 되네 ? 10
```

- bigLocationCheck

```
static public void bigLocationCheck () {
    int[] numArr = new int[arrLength];

    for (int i = arrLength - 1; i >= 0; i--) {
        numArr[i] = bigNum.divide(
                new BigInteger(
                        String.valueOf(
                                //Math.pow(BASE, i)
                                TEN.pow(i)
                        )
                )
        ).mod(
                TEN
        ).intValue();

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

    System.out.println("실제 집어넣은 값: " + bigNum);
}
```

계산된 arrLength 크기를 가지는 numArr 배열을 생성한다.
bigNum = 45678911234
배열의 길이 arrLength = 11
I의 초기값 = arrLength – 1 -> 1씩 감소
bigNum을 10^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.
ex) 45678911234 / 10^10 = 4
    4 % 10 = 4

```
numArr[10] = 4
numArr[9] = 5
numArr[8] = 6
numArr[7] = 7
numArr[6] = 8
numArr[5] = 9
numArr[4] = 1
numArr[3] = 1
numArr[2] = 2
numArr[1] = 3
numArr[0] = 4
실제 집어넣은 값: 45678911234
```

7. 회사에 직원이 7명이 있습니다.
   모두 입사동기로 3500만원으로 시작하였다고 합니다.
   각자의 연봉 인상률이 매년 1 ~ 10%라고 합니다.
   이들이 5년후에 받을 연봉에 대한 시뮬레이션을 랜덤하게 돌려봅시다.

[Bank3Ans7ClassTest (main) - EmployeeManager - Employee]

- Bank3Ans7ClassTest (main)

EmployeeManager 클래스의 생성자 호출해서 em이라는 객체 생성
이 때 생성자에 firstpay와 num값을 넘겨준다,
(firstpay: 초봉, num = 직원 수)

year값을 넘겨주며 EmployeeManager 클래스의 paymentSimulation 메소드를 호출한다.

- EmployeeManager

1. 배열 선언

```
Employee[] empArr;
```

직원들의 연봉이 기록되는 배열 empArr

2. 생성자

```
public EmployeeManager (final float firstPay, final int num) {
    empArr = new Employee[num];

    for (int i = 0; i < num; i++) {
        empArr[i] = new Employee(firstPay);
    }
}
```

main에서 firstpay와 num을 받아온다.
num의 크기를 가지는 배열 empArr을 생성한다.

for문을 통해 직원 수인 num만큼 반복을 수행한다.
반복문 내부에서는 배열의 각 인덱스에 순차적으로 Employee 형식의 객체를 생성한다.
Employee 클래스의 생성자에 넘겨주는 값은 firstpay

3. paymentSimulation

```
public void paymentSimulation (int year) {
    // 어차피 연수를 입력 받으므로
    // 직원들의 전체 랜덤한 증가율을 미리 할당한 이후
    // 그다음에 7년치를 계산하도록 한다.
    allocEmpIncRatio(year);
    calcPayment(year);
}
```

main에서 호출되는 EmployeeManager 클래스의 메소드로
호출될 때 main으로부터 year 받아온다.
allocEmpIncRatio 메소드와 calcPayment 메소드를 호출한다.

4. allocEmpIncRatio

```
public void allocEmpIncRatio (int year) {
    for (int i = 0; i < empArr.length; i++) {
        empArr[i].allocIncRatio(year);
    }
}
```

for문을 통해 empArr배열의 길이만큼 반복을 수행한다.
반복문 내부에서는 각 인덱스에 할당된 객체들의 allocIncRatio 메소드를 호출한다.
Employee 클래스의 allocIncRatio 메소드에 넘겨주는 값은 year

5. calcPayment

```
public void calcPayment (int year) {
    for (int i = 0; i < empArr.length; i++) {
        empArr[i].calcPayment(year);
    }
}
```

for문을 통해 empArr배열의 길이만큼 반복을 수행한다.
반복문 내부에서는 각 인덱스에 할당된 객체들의 calcPayment 메소드를 호출한다.
Employee 클래스의 calcPayment 메소드에 넘겨주는 값은 year

- Employee

1. 변수 선언

```
float payment;
// 1 ~ 10% --> 1.00 ~ 10.00
float[] incRatio;

final int MIN = 100;
final int MAX = 1000;

final float BIAS = 100.0f;

int range;
```

연봉을 저장할 payment
각 객체의 5년치 연봉인상률에 대한 정보를 가진 배열 incRatio

연봉인상률 계산에 사용할 변수
MIN, MAX, MIAS, range

2. 생성자

```
public Employee (final float firstPay) {
    payment = firstPay;

    range = MAX - MIN + 1;
}
```

객체 생성 시 EmployeeManager 클래스로부터 firstpay 값을 받아와
각 객체의 payment 변수에 대입한다.
연봉인상률의 범위를 계산한다.

3. allocIncRatio

```
public void allocIncRatio (int year) {
    incRatio = new float[year];

    for (int i = 0; i < year; i++) {
        incRatio[i] = ((int) (Math.random() * range + MIN)) / BIAS;
        // System.out.printf("incRatio[%d] = %.2f\n", i, incRatio[i]);
    }
}
```

EmployeeManager 클래스로부터 year 값을 받아와
year만큼의 크기를 가지는 배열 incRatio를 생성한다.
for문을 통해 year만큼 반복을 수행한다.
반복문 내부에서는 범위(1~10%) 안의 난수를 발생시켜 incRatio의 인덱스에 순차적으로 저장한다.

4. calcPayment

```
public void calcPayment (int year) {
    for (int i = 0; i < year; i++) {
        // payment = payment + payment * incRatio[i] / BIAS;
        payment += (payment * incRatio[i] / BIAS);
    }

    System.out.printf("5년후의 연봉 = %d\n", (int)payment);
}
```

EmployeeManager 클래스로부터 year 값을 받아온다.
for문을 통해 year만큼 반복을 수행한다.
반복문 내부에서는 5년치 연봉인상률을 가진 incRatio배열의 인덱스를 통해 각 연차에 따른 연봉을 계산한다.
payment = payment + (payment * incRatio[i] / BIAS);
반복이 종료되면 payment에는 5년 후의 연봉이 저장되며 이 값을 출력한다.

```
5년후의 연봉 = 50116148
5년후의 연봉 = 44773184
5년후의 연봉 = 50276036
5년후의 연봉 = 39306596
5년후의 연봉 = 47121220
5년후의 연봉 = 45116556
5년후의 연봉 = 47740900
```

8. 2명이 주사위 게임을 한다. (배열 활용)
   주사위는 각자 2개씩 굴릴 수 있다.
   처음 주사위를 굴렸을때 결과가 짝수라면 한 번 더 돌릴 수 있다.
   (2, 4, 6, 8, 10, 12)
   한 번 더 돌리는 주사위는 특수 스킬을 가지고 있다.
   (특수 스킬 주사위는 1번만 굴린다)
   이 특수 스킬들은 1, 3, 4, 6에서 동작한다.
   1번의 경우 상대방의 주사위 눈금을 2 떨군다.
   3번의 경우 다 같이 -6을 적용한다. (결과는 0 이하로 떨어지지 않는다 - 무승부 노리기)
   4번의 경우 그냥 패배 처리한다.
   6번의 경우 모든 상대방에게 3을 뺏어서 내거에 3을 더한다.
   2번, 5번은 그냥 특수 스킬이 동작하지 않고 단순히 더해진다.

[Bank3Ans8ClassTest (main) - GameManager - Player - Dice]

- Bank3Ans8ClassTest (main)

```
public class Bank3Ans8ClassTest {
    public static void main(String[] args) {
        // 주사위는 각자 2개씩 굴릴 수 있다.
        // 처음 주사위를 굴렸을때 결과가 짝수라면 한 번 더 돌릴 수 있다.
        // (2, 4, 6, 8, 10, 12)

        // 1. Player 클래스가 필요함
        // 2. Dice 클래스가 필요함
        // 3. GameManager 클래스가 필요함
        GameManager gm = new GameManager( playerNum: 2,  diceCnt: 2); // 사용자 수, 사용하는 주사위 수

        gm.startGame();
    }
}
```

GameManager 클래스의 생성자를 호출해 객체 gm을 생성한다.
생성자 호출 시 사용자 수를 의미하는 playerNum과 사용하는 주사위 수를 의미하는 diceCnt값을 넘겨준다.

gm 객체의 startGame 메소드를 호출한다.

- GameManager

1. 변수 선언

```
Player[] players;
int playerNum;
```

각 사용자(Player 객체)들이 할당될 Player 형식의 배열 players 선언
사용자의 수를 의미하는 playerNum

2. 생성자

```
public GameManager (final int playerNum, final int diceCnt) {
    this.playerNum = playerNum;
    players = new Player[playerNum];

    for (int i = 0; i < playerNum; i++) {
        players[i] = new Player(diceCnt);
    }
}
```

main으로부터 사용자 수(playerNum)와 사용하는 주사위 수(diceCnt)를 받아온다.
GameManger 클래스의 playerNum 변수에 받아온 playerNum을 대입한다.
playerNum의 크기를 가지는 Player 형식의 객체배열 players를 생성한다.

for문을 통해 playerNum만큼 반복을 수행한다.
매 루프마다 객체배열 players의 각 인덱스에서 Player 클래스의 생성자를 호출해 실제 객체를 생성한다.
이 때 diceCnt 값을 넘겨준다.

3. startGame

```
public void startGame () {
    System.out.println("주사위 게임 시작!");

    rollPlayerDice();
    checkSum();
    rollSpecialDice();
    applySkillEffect();
}
```

GameManager 클래스의 메소드들을 호출한다.

4. rollPlayerDice

```
public void rollPlayerDice () {
    for (int i = 0; i < playerNum; i++) {
        players[i].rollEveryDice();
    }
}
```

for문을 통해 playerNum만큼 반복을 수행한다.
매 루프마다 players 배열에 생성된 각 객체의 rollEveryDice 메소드를 호출한다.

5. rollSpecialDice

```
public void rollSpecialDice () {
    for (int i = 0; i < playerNum; i++) {
        if (players[i].checkSpecialDice()) {
            players[i].rollSpecialDice();
        }
    }
}
```

for문을 통해 playerNum만큼 반복을 수행한다.
매 루프마다 players 배열에 생성된 각 객체의 checkSpecialDice 메소드를 호출해 특수주사위 발동 여부를 판단한다.
Player 클래스의 checkSpecialDice 메소드는 boolean값을 리턴한다.
리턴값이 true라면 객체의 rollSpecialDice 메소드를 호출한다.

6. applySkillEffect

```
public void applySkillEffect () {
    int tmp;
    for (int i = 0; i < playerNum; i++) {
        if (players[i].isGetSpecial()) {
```

특수주사위 값을 저장할 변수 tmp를 선언한다.
for문을 통해 playerNum만큼 반복을 수행한다.
players 배열의 각 객체에서 isGetSpecial 메소드를 호출해 특수주사위 발동조건인지 확인한다.

```
switch (tmp = players[i].getSpecialDiceNum()) {
    case 1:
        // 1번의 경우 상대방의 주사위 눈금을 2 떨군다.
        System.out.println("1번 - 상대 눈금을 2떨굼");
        for (int j = 0; j < playerNum; j++) {
            if (i == j) {
                continue;
            }

            players[j].operateDice( num: -2);
        }
        break;
```

각 특수주사위 값에 따른 별도의 처리를 위해 switch문을 사용한다.
switch의 기준값은 tmp로, 매 루프마다 players 배열의 각 객체에서 getSpecialDiceNum 메소드를 호출해 리턴값을 tmp에 대입한다.

- 특수주사위 값이 1인 경우 : 상대방의 눈금을 2 떨군다.
for문을 통해 playerNum만큼 반복을 수행한다.

상위 루프인 I는 현재 플레이어를 의미한다.
switch 내부에 위치한 하위 루프인 j의 값이 I와 같아질 때는 continue를 통해 skip한다.
I의 값과 j의 값과 다를 때, players[j]는 상대 플레이어를 의미하므로 해당 객체의 operateDice 메소드를 호출한다. 이 때 num값(-2)을 전달하며 전달된 num만큼 해당 객체의 눈금값에 증감이 이루어진다.

```
case 3:
    // 3번의 경우 다 같이 -6을 적용한다.
    System.out.println("3번 - 다같이 6떨굼");
    for (int j = 0; j < playerNum; j++) {
        players[j].operateDice( num: -6);
    }
    break;
```

- 특수주사위 값이 3인 경우 : 모두의 눈금을 6 떨군다.
for문을 통해 playerNum만큼 반복을 수행한다.
반복문 내부에서는 각 객체의 operateDice 메소드를 호출한다.
이 때 num값으로 -6을 전달해 전달된 num만큼 모든 객체의 눈금값에 증감이 이루어진다.

- 특수주사위 값이 4인 경우 : 해당 플레이어 패배

```
case 4:
    // 4번의 경우 그냥 패배
    System.out.println("4번 - 잘가");
    players[i].operateDice( num: 4444);
    break;
```

현재 플레이어 객체 players[i]의 opearateDice 메소드를 호출한다.
이 때 num값으로 4444를 전달해 패배시킨다.

- 특수주사위 값이 6인 경우 : 모든 상대방에게 3을 뺏어서 내 눈금에 더한다.

```
case 6:
    // 6번의 경우 모든 상대방에게 3을 뺏어서 내거에 3을 더한다.
    System.out.println("6번 - 상대편 3을 뺏어서 내쪽으로 3을 땡겨옴");
    for (int j = 0; j < playerNum; j++) {
        if (i == j) {
            continue;
        }

        players[j].operateDice( num: -3);
        players[i].operateDice( num: 3);
    }
    break;
```

for문을 통해 playerNum만큼 반복을 수행한다.

상위 루프인 I는 현재 플레이어를 의미한다.
switch 내부에 위치한 하위 루프인 j의 값이 I와 같아질 때는 continue를 통해 skip한다.
I의 값과 j의 값과 다를 때,
players[j]는 상대 플레이어를 의미하므로 해당 객체의 operateDice 메소드를 호출한다.
이 때 num값(-2)을 전달하며 전달된 num만큼 해당 객체의 눈금값에 증감이 이루어진다.
그리고 현재 플레이어인 players[i] 객체의 operateDice 메소드를 호출한다.
이 때 num값으로 3을 전달하며 전달된 3만큼 현재 플레이어의 눈금값에 증감이 이루어진다.

- 특수주사위 값이 2나 5일 때 : 단순히 현재 눈금합에 특수주사위 눈금합을 더한다.

```
default:
    System.out.println("디폴트 2, 5!!!");
    players[i].operateDice(tmp);
    break;
```

현재 플레이어 객체 players[i]의 opearateDice 메소드를 호출한다.
이 때 num값으로 tmp를 전달해 해당 객체의 눈금값에 tmp만큼의 증감이 이루어진다.

7. checkSum

```
public void checkSum () {
    for (int i = 0; i < playerNum; i++) {
        System.out.printf("플레이어 %d - %d\n", i, players[i].getSum());
    }
}
```

for문을 통해 playerNum만큼 반복을 수행한다.
players 배열에 저장된 각 객체들의 getSum메소드를 호출해 출력시켜
눈금합을 확인한다.

- Player

1. 변수 선언

```
int diceCnt;
Dice[] diceArr;
Dice special;

int sum;
boolean getSpecial;
```

사용하는 주사위 수를 의미하는 diceCnt
Dice 형식의 배열 diceArr 선언
특수주사위에 해당하는 Dice 형식의 객체 special 선언
주사위 합을 의미하는 sum
특수주사위 발동조건을 판별하기 위한 getSpecial

2. 생성자

```
public Player (final int diceCnt) {
    this.diceCnt = diceCnt;
    diceArr = new Dice[diceCnt];

    for (int i = 0; i < diceCnt; i++) {
        diceArr[i] = new Dice();
    }

    special = new Dice();
}
```

GameManager 클래스로부터 diceCnt값을 받아와 Player 클래스의 diceCnt에 대입한다.
diceCnt만큼의 크기를 가지는 Dice타입의 배열 diceArr를 생성한다.
for문을 통해 diceCnt만큼 반복을 수행한다.
diceArr 배열의 각 인덱스에서 Dice 생성자를 호출한다.
for문을 통해 diceCnt만큼의 Dice 객체가 할당된 후,
특수주사위에 해당하는 Dice타입의 객체 special를 생성한다.

3. rollEveryDice

```
public void rollEveryDice () {
    sum = 0;

    for (int i = 0; i < diceCnt; i++) {
        diceArr[i].rollDice();
        sum += diceArr[i].getDiceNum();
    }
}
```

sum을 0으로 초기화한다.
for문을 통해 diceCnt만큼 반복을 수행한다.
반복문 내부의 각 인덱스에 위치한 Dice 타입의 객체 diceArr[i]들은 Dice 클래스의
rollDice 메소드를 호출한 뒤 getDiceNum 메소드를 호출해 리턴값을 sum에 더한다.

4. getSum

```
public int getSum () { return sum; }
```

rollEverDice 메소드를 통해 구해진 Sum값을 GameManager 클래스에 반환한다.

5. checkSpecialDice

```
public boolean checkSpecialDice () {
    if (sum % 2 == 0) {
        getSpecial = true;
        return getSpecial;
    } else {
        getSpecial = false;
        return getSpecial;
    }
}
```

특수주사위 발동 조건에 해당하는지 확인하기 위해
rollEverDice 메소드를 통해 구해진 Sum값에 %2 연산을 수행한다.
연산의 결과가 0이라면 짝수에 해당하므로 getSpecial값은 true가 되어
gameManager 클래스에 반환된다.
연산의 결과가 0이 아니라면 홀수에 해당하므로 getSpecial값은 false가 되어
gameManager 클래스에 반환된다.

6. rollSpecialDice

```
public void rollSpecialDice () { special.rollDice(); }
```

특수주사위를 의미하는 special 객체의 rollDice 메소드를 호출한다.

7. isGetSpecial

```
public boolean isGetSpecial() { return getSpecial; }
```

GameManager 클래스는 특수주사위 값을 반영하기 전에 특수주사위 발동조건을
다시 확인한다. 이 때 호출되는 메소드로 checkSpecialDice 메소드를 통해
정해진 getSpecial 값을 GameManager 클래스에 반환한다.

8. getSpecialDiceNum

```
public int getSpecialDiceNum () { return special.getDiceNum(); }
```

특수주사위를 의미하는 special 객체의 getDiceNum 메소드를 호출해 결과값을
GameManger 클래스에 반환한다.

9. operateDice

```
public void operateDice (int num) {
    sum += num;

    if (sum < 0) {
        sum = 0;
    }
}
```

sum값에 대한 증감을 처리하는 메소드로 GamaManager 클래스로부터
num값을 받아와 sum에 더한다.
num값으로 음수를 받아올 수도 있다.
이 때 음수로 인해 sum값이 0보다 작아지는 현상을 방지하기 위해
sum이 0보다 작을 때는 sum을 0으로 처리한다.

- Dice

1. 변수 선언

```
final int MAX = 6;
final int MIN = 1;

int range;
int diceNum;
```

주사위 값의 범위 (1 ~ 6)을 지정하기 위한 MAX, MIN, range
주사위 값을 의미하는 diceNum

2. 생성자

```
public Dice () {
    //System.out.println("나는 Dice 클래스의 기본 생성자!");
    range = MAX - MIN + 1;
}
```

Dice 타입의 객체가 생성되면 해당 객체의 주사위 값의 범위를 계산한다.

3. rollDice

```
public void rollDice () { diceNum = (int) (Math.random() * range + MIN); }
```

1~6 사이의 난수를 발생시켜 주사위 값인 diceNum에 대입한다.

4. getDIceNum

```
public int getDiceNum () { return diceNum; }
```

rollDice 메소드를 통해 구해진 diceNum을 Player 클래스에 반환한다.

```
주사위 게임 시작!
플레이어 0 - 10
플레이어 1 - 8
디폴트 2, 5!!!
6번 - 상대편 3을 뺏어서 내쪽으로 3을 땡겨옴
```