2022.1.4 인지연 클래스 – 생성자 개념

문제 1. 정수형 변수 1개, float 변수 2개, double 변수 3개에 아무 값이나 할당하고 출력합니다.

생성자 만드는 방법
1. public을 적는다.
2. 클래스 이름과 동일한 이름을 작성한다.
3. 매서드 작성하듯이 입력으로 사용할 정보들을 입력하도록 한다.
4. 중괄호 내부에 이 매서드(생성자)가 구동할 작업을 작성한다.

*** 흥미로운 부분이라면 생성자는 리턴 타입이 없다!!! 즉 void도 쓰지 않는다.

```
public class MyCustomDataTypeTest {
    public static void main(String[] args) {
        // 정수형 변수 1개, float 변수 2개, double 변수 3개에 아무 값이나 할당하고 출력합니다.
        // 이 문제는 생성자 + 함수 덮어쓰기를 함께 활용하면 효율적으로 만들 수 있다.

        MyCustomDataType mcdt = new MyCustomDataType(1, 2, 3);
        MyCustomDataType mcdt2 = new MyCustomDataType(1, 2f, 3.0);
        MyCustomDataType mcdt3 = new MyCustomDataType();
```

1) 생성자 (int,int,int)

```
public MyCustomDataType (int intNum, int floatNum, int doubleNum) {
    System.out.println("나는 (int, int, int) 생성자!");

    intArr = new int[intNum];
    floatArr = new float[floatNum];
    doubleArr = new double[doubleNum];

    setRange();
}
```

2) 생성자 (int, float, double)

```
public MyCustomDataType (int intNum, float floatNum, double
doubleNum) {
    System.out.println("나는 (int, float, double) 생성자!");
}
```

3) 생성자 (내용 없음)

```
public MyCustomDataType () {
    System.out.println("나는 기본 생성자!");
}
```

```
나는 (int, int, int) 생성자!
나는 (int, float, double) 생성자!
나는 기본 생성자!
```

아래의 뜻은 무엇이냐?

1. MyCustomDataType클래스 객체 생성
2. 생성자에 (int, int, int)작성
3. 그 객체 안에있는 allocRandom 매소드 호출
4. System.out.println(mcdt)이렇게 쓰면
mcdt의 toString값을 도출할 수 있다.

```
public class MyCustomDataTypeTest {
    public static void main(String[] args) {

        MyCustomDataType mcdt = new
1)      MyCustomDataType(1, 2, 3);

2)      mcdt.allocRandom();
        System.out.println(mcdt);
```

1)

```
public MyCustomDataType (int intNum, int floatNum, int doubleNum) {
    System.out.println("나는 (int, int, int) 생성자!");

    intArr = new int[intNum];
    floatArr = new float[floatNum];
    doubleArr = new double[doubleNum];

    setRange();
}
```

요기서 배열 값 받음

요것도 주의깊게 봐야한다.
아까 본 것 중에 하나였는데, 메인 매소드에서
값을 입력하면 각 배열 크기가 입력되는 것 이였음!!

2)

```
public void allocRandom () {
    allocIntRandom();
    allocFloatRandom();
    allocDoubleRandom();
}
```

allocRandom들어갔더니,
각각 랜덤 매소드가 또있네,
보러가자

```
public void setRange () {
    setIntRange();
    setRealRange();
}
```

미쳤다..ㅎ
요게있네..

```
public void allocIntRandom () {
    // length는 실제 배열의 길이를 구해올 수 있음
    for (int i = 0; i < intArr.length; i++) {
        intArr[i] = (int) (Math.random() * intRange + INT_MIN);
    }
}

public void allocFloatRandom () {
    for (int i = 0; i < floatArr.length; i++) {
        // (0.0 ~ 950) + (-4.75f)
        //float tmp = ((int) (Math.random() * floatRange)) / BIAS +
FLOAT_MIN;

        float tmp = (int) (Math.random() * floatRange);
        //System.out.printf("tmp = %f\n", tmp);

        tmp /= BIAS;
        //System.out.printf("tmp = %f\n", tmp);

        floatArr[i] = tmp + FLOAT_MIN;
    }
}

public void allocDoubleRandom () {
    for (int i = 0; i < doubleArr.length; i++) {
        double tmp = (int) (Math.random() * floatRange);
        tmp /= BIAS;

        doubleArr[i] = tmp + FLOAT_MIN;
    }
}
```

```
public void setIntRange () {
    intRange = INT_MAX - INT_MIN + 1;
}

public void setRealRange () {
    // 0.0 ~ 1.0 미만 - Math.random()
    // 4.75 + 4.75 -> 9.5
    // 0.0 ~ 9.5 미만 + 0.1
    // 0.1 ~ 9.6 미만 (9.5xx)
    // 그래서 최종적으로 ===> (-475 ~ 475) / 100
    // floatRange = 951
    floatRange = FLOAT_MAX * BIAS - FLOAT_MIN * BIAS + 1;
}
```

각 값의 랜덤기준으로 정했다.
이 값을 통해서,
alloc~random문에 ~range를 입력했네

각각 int형 / float형 /double형 랜덤문을 만들어놓았다.
(이 안에서 어떤 역할을 하는지는 이따 자세히 살펴보자
우선은 이렇게 루트가 간다를 보는것 먼저!)

그러면 리턴하는 것이 따로 없으니, 우선 이 메소드가
있다는 것으로 끝낸다.

그러면 각각의 값을 낼 수 있겠지?

보니까 각 배열 값에 랜덤 값을 for문으로 값을 저장하고있다.
이걸 메인매소드에 호출하면서 그 값을 가져오게 한다.

메인매소드로 가보자.
이건 앞서 말했듯, mcdt(MyCustomDataType) 객체의 toString 값을 출력해준다.

MyCustomDataType의 toString 구경가보자

```
public class MyCustomDataTypeTest {
    public static void main(String[] args) {
        MyCustomDataType mcdt = new MyCustomDataType(1, 2, 3);
        MyCustomDataType mcdt2 = new MyCustomDataType(1, 2f, 3.0);
        MyCustomDataType mcdt3 = new MyCustomDataType();

        mcdt.allocRandom();
  3)    System.out.println(mcdt);
```

보다싶이, 우리가 보고싶은 변수 값을 지정할 수 있음.
우리는 intArr / floatArr / doubleArr값을 볼 수 있다. 저 값은 뭐지? 어딨지? 살펴보자.

```
@Override
public String toString() {
    return "MyCustomDataType{" +
            "intArr=" + Arrays.toString(intArr) +
            ", floatArr=" + Arrays.toString(floatArr) +
            ", doubleArr=" + Arrays.toString(doubleArr) +
            '}';
}
```

```
public void allocIntRandom () {
    // length는 실제 배열의 길이를 구해올 수 있음
     for (int i = 0; i < intArr.length; i++) {
        intArr[i] = (int) (Math.random() * intRange + INT_MIN);
    }
}

public void allocFloatRandom () {
    for (int i = 0; i < floatArr.length; i++) {
        // (0.0 ~ 950) + (-4.75f)
        //float tmp = ((int) (Math.random() * floatRange)) / BIAS +
FLOAT_MIN;

        float tmp = (int) (Math.random() * floatRange);
        //System.out.printf("tmp = %f\n", tmp);

        tmp /= BIAS;
        //System.out.printf("tmp = %f\n", tmp);

        floatArr[i] = tmp + FLOAT_MIN;
    }
}

public void allocDoubleRandom () {
    for (int i = 0; i < doubleArr.length; i++) {
        double tmp = (int) (Math.random() * floatRange);
        tmp /= BIAS;

        doubleArr[i] = tmp + FLOAT_MIN;
    }
}
```

아,. 아까 우리가 봤던거네.
보면 intArr배열 값을 출력하는것 같네.
아까 보니까
int ->1
float -> 2
double -> 3개의 크기를 가지고있었고,
랜덤 값으로 각 배열 주소에 저장했었음.

출력 값 확인해보자.

```
mcdt.allocRandom();
System.out.println(mcdt);
```

메인매소드의 저 출력값은 결국
Int형 랜덤 1개 / float형 랜덤 2개 /
Double형 랜덤 3개가 출력된다!
배열의 값을 for문 통해 하나씩 넣었기 때문

```
MyCustomDataType{intArr=[37], floatArr=[1.4699998, -4.5], doubleArr=[-2.07, 0.75, -3.62]}
```

제일 어려운거 나왔다. 빨간 네모 봐랏.
생성자에 2 그리고 어떤게 나왔는데.. 그 안에 이상한 글자가 있다.
MyCustomDataType 안에 있는 INT_PROC 변수를 쓴 것 같다.

```
public class MyCustomDataTypeTest {
    public static void main(String[] args) {
        MyCustomDataType mcdt = new MyCustomDataType(1, 2, 3);
        MyCustomDataType mcdt2 = new MyCustomDataType(1, 2f, 3.0);
        MyCustomDataType mcdt3 = new MyCustomDataType();

        mcdt.allocRandom();
        System.out.println(mcdt);

        MyCustomDataType mcdt4 = new MyCustomDataType(2, MyCustomDataType.INT_PROC);
        mcdt4.allocIntRandom();
        System.out.println(mcdt4);

        MyCustomDataType mcdt5 = new MyCustomDataType(2, MyCustomDataType.FLOAT_PROC);
        mcdt5.allocFloatRandom();
        System.out.println(mcdt5);

        MyCustomDataType mcdt6 = new MyCustomDataType(2, MyCustomDataType.DOUBLE_PROC);
        mcdt6.allocDoubleRandom();
        System.out.println(mcdt6);
    }
}
```

처음에 헷갈렸던 것, MyCustomDataType 매소드와 decisionAlloc 매소드가 (int, int)형을 출력 받는데
어째서 MyCustomDataType로 값이 출력될 수 있을까? 아주 바보같은 생각.
당연히 메인매소드 호출값이 **생성자**였기 때문에, 생성자인 MyCustomDataType 매소드로 들어가서 출력 받는 것이다.

밑에를 봐라. 그러면 이상한게 있다. DECISION 이게 뭐냐 하고 보니까
메인매소드에서 (MyCustomDataType.INT_PROC) 이렇게 써져 있었다.
INT_PROC의 값은 1이라는 상수를 적어놓았었다.(다음페이지에서 설명)

```
public MyCustomDataType (int intNum, final int DECISION) {
    System.out.println("나는 (int, int) 생성자!");

    System.out.println("DECISION: " + DECISION);
    decisionAlloc(intNum, DECISION);
```

그럼 이 값은
"DICESION: 1" 값이 나오겠지?
INT_PROC값이 상수 : 1이다.

decisionAlloc매소드를 그럼 가보자.
이 매소드 값의 ()가 뭐라고 되어있는지 보자. (배열크기 / decision값)이다.
Switch를 적용해서 각자 쓴 값(~_PROC→ 1,2,3)으로 값 도출하면서 필요한 랜덤값은 set~Range매소드 사용.
마지막에 break를 해서 다른 case에는 값이 나오지 않도록 했다.

```
public void decisionAlloc(int arrNum, final int DECISION) {
    switch (DECISION) {
        case INT_PROC:
            intArr = new int[arrNum];
            setIntRange();
            break;

        case FLOAT_PROC:
            floatArr = new float[arrNum];
            setRealRange();
            break;

        case DOUBLE_PROC:
            doubleArr = new double[arrNum];
            setRealRange();
            break;

        default:
            System.out.println("올바른 값을 입력하세요!");
            break;
    }
}
```

```
public void setIntRange () {
    intRange = INT_MAX - INT_MIN + 1;
}

public void setRealRange () {
    // 0.0 ~ 1.0 미만 - Math.random()
    // 4.75 + 4.75 -> 9.5
    // 0.0 ~ 9.5 미만 + 0.1
    // 0.1 ~ 9.6 미만 (9.5xx)
    // 그래서 최종적으로 ===> (-475 ~ 475) / 100
    // floatRange = 951
    floatRange = FLOAT_MAX * BIAS - FLOAT_MIN * BIAS + 1;
}
```

이 두개는 어차피 실수라서
랜덤 범위가 같다.

우리가 정리했던 변수들

```
int[] intArr;
float[] floatArr;
double[] doubleArr;

int intRange;
float floatRange;

final int INT_MAX = 50;
final int INT_MIN = 30;

final float FLOAT_MAX = 4.75f;
final float FLOAT_MIN = -4.75f;
final int BIAS = 100;

static final int INT_PROC = 1;
static final int FLOAT_PROC = 2;
static final int DOUBLE_PROC = 3;
```

드디어 이걸 정리할 때가 됐다..
데이터 타입 단위로 체크해보자.

```
public void allocIntRandom () {
    // length는 실제 배열의 길이를 구해올 수 있음
    for (int i = 0; i < intArr.length; i++) {
        intArr[i] = (int) (Math.random() * intRange + INT_MIN);
    }
}

public void allocFloatRandom () {
    for (int i = 0; i < floatArr.length; i++) {
        // (0.0 ~ 950) + (-4.75f)
        //float tmp = ((int) (Math.random() * floatRange)) / BIAS + FLOAT_MIN;

        float tmp = (int) (Math.random() * floatRange);
        //System.out.printf("tmp = %f\n", tmp);

        tmp /= BIAS;
        //System.out.printf("tmp = %f\n", tmp);

        floatArr[i] = tmp + FLOAT_MIN;
    }
}

public void allocDoubleRandom () {
    for (int i = 0; i < doubleArr.length; i++) {
        double tmp = (int) (Math.random() * floatRange);
        tmp /= BIAS;

        doubleArr[i] = tmp + FLOAT_MIN;
    }
}
```

```
public void allocIntRandom () {
    // length는 실제 배열의 길이를 구해올 수 있음
    for (int i = 0; i < intArr.length; i++) {
        intArr[i] = (int) (Math.random() * intRange + INT_MIN);
    }
}
```

```
public void setIntRange () {
    intRange = INT_MAX - INT_MIN + 1;
}

public void setRealRange () {
    // 0.0 ~ 1.0 미만 - Math.random()
    // 4.75 + 4.75 -> 9.5
    // 0.0 ~ 9.5 미만 + 0.1
    // 0.1 ~ 9.6 미만 (9.5xx)
    // 그래서 최종적으로 ===> (-475 ~ 475) / 100
    // floatRange = 951
    floatRange = FLOAT_MAX * BIAS - FLOAT_MIN * BIAS + 1;
}
```

(4.75 * 100) – (-4.75 * 100) + 1 = 951

Int형은 간단하다.
30~50 사이에 랜덤 값 도출하는것.
intRange값을 따로 빼서 가독성이 더 좋게 만들었다.

```
public void allocFloatRandom () {
    for (int i = 0; i < floatArr.length; i++) {
        // (0.0 ~ 950) + (-4.75f)
        //float tmp = ((int) (Math.random() * floatRange)) / BIAS + FLOAT_MIN;

        float tmp = (int) (Math.random() * floatRange);

        tmp /= BIAS;

        floatArr[i] = tmp + FLOAT_MIN;
    }
}
```

0.0~0.999 * 951 = 0 ~ 950.999...

int형 변환 : 0~950 을 float형으로 가지고 있는다.

0 ~ 950 / 100 = 0 ~ 9.50000

0 ~ 9.50000 + (-4.75) = -4.75 ~ +4.75

실수는 float만 해도 double이랑 같이 동일하게 때문에 이것만 보자. 복잡해 보이지만 알고 보면 괜츈
-4.75~+4.75 랜덤 값 구하는 내용이다.
선생님의 랜덤 계산 공식 : setRealRange에 있음.
원래 랜덤의 범위는 0~0.999.. 이다.
기존 공식처럼 max – min 을 한다면 4.75 – (-4.75) = 9.5가 된다.
여기서 0.1을 더하면 0.1 ~ 9.6으로 9.5999까지 나오게 될거라 이건 영 좋지 않다.
예전에 선생님이 했던 방식으로 -475 ~ +475 의 값을 /100을 해서 문제없이 값이 나오게 된다.

```java
public class MyCustomDataTypeTest {
    public static void main(String[] args) {
        // 정수형 변수 1개, float 변수 2개, double 변수 3개에 아무 값이나 할당하고 출력합니다.
        // 이 문제는 생성자 + 함수 덮어쓰기를 함께 활용하면 효율적으로 만들 수 있다.

        // 첫번째는 정수형 개수
         // 두번째는 float 개수
         // 세번째는 double 개수를 입력하도록 설정

          // 여태까지 객체화하는 작업이 new 클래스명()으로 알고 있었을 것이다.
        // 실제로 객체화 하는 작업이 맞고 이를 수행할때 생성자라는 녀석이 동작하게 된다.
        // 생성자는 아래와 같이 매서드처럼 입력을 전달 받을 수 있다.
        MyCustomDataType mcdt = new MyCustomDataType(1, 2, 3);
        MyCustomDataType mcdt2 = new MyCustomDataType(1, 2f, 3.0);
        MyCustomDataType mcdt3 = new MyCustomDataType();

        mcdt.allocRandom();
        System.out.println(mcdt);

        MyCustomDataType mcdt4 = new MyCustomDataType(2, MyCustomDataType.INT_PROC);
        mcdt4.allocIntRandom();
        System.out.println(mcdt4);

        MyCustomDataType mcdt5 = new MyCustomDataType(2, MyCustomDataType.FLOAT_PROC);
        mcdt5.allocFloatRandom();
        System.out.println(mcdt5);

        MyCustomDataType mcdt6 = new MyCustomDataType(2, MyCustomDataType.DOUBLE_PROC);
        mcdt6.allocDoubleRandom();
        System.out.println(mcdt6);
    }
}
```

위의 출력값은 아래와 같다.
주의해야할 점 : ~_PROC라고 되어있는 것들은
INT_PROC -> INT만 출력 // FLOAT_PROC → FLOAT만 출력.. 이런식으로 된다.
그리고 앞에 2는 배열의 크기를 나타내기 때문에 각 값이 두개씩 랜덤으로 나옴

```
나는 (int, int, int) 생성자!
나는 (int, float, double) 생성자!
나는 기본 생성자!
MyCustomDataType{intArr=[37], floatArr=[1.4699998, -4.5], doubleArr=[-2.07, 0.75, -3.62]}
나는 (int, int) 생성자!
DECISION: 1
MyCustomDataType{intArr=[39, 36], floatArr=null, doubleArr=null}
나는 (int, int) 생성자!
DECISION: 2
MyCustomDataType{intArr=null, floatArr=[2.46, 0.52], doubleArr=null}
나는 (int, int) 생성자!
DECISION: 3
MyCustomDataType{intArr=null, floatArr=null, doubleArr=[-2.33, 0.9000000000000004]}
```

클래스 – 문제 2. 주사위 2개를 굴려서 눈금의 합을 출력해봅시다.

아까 문제보다는 조금 간단해보이지만, 이번에는 메인클래스 포함 클래스는 총 3개다.
왜 클래스가 3개로 나눠서 진행되는지 한번 살펴보도록 하자

```
public class DiceGameTest {
    public static void main(String[] args) {
        // 주사위 2개를 굴려서 눈금의 합을 출력해봅시다.
        // 주사위의 개수 자체가 확장이 가능함
          DiceManager dm = new DiceManager(2);

        dm.playDiceGame();
        System.out.println(dm);
    }
}
```

메인클래스에서 한일

1) DiceManager 생성자 만들면서 2를 넣었다. → 2개의 배열 크기를 만들면서 값을 받을 수 있게 했다.
2) dm.playDiceGame(); 으로 매소드 호출했다.
3) System.out.println(dm);으로 toString호출했다.

```
public class DiceManager {
    int diceNum;

    Dice[] diceArr;
    int sum;

1)  public DiceManager (int diceNum) {
        sum = 0;
        this.diceNum = diceNum;

        diceArr = new Dice[diceNum];
    }

    public void checkWin (int sum) {
        if (sum % 2 == 0) {
            System.out.println("당첨입니다!!!");
        } else {
            System.out.println("올땐 쉬웠지 갈땐 손모가지닷!");
        }
    }

2)  public void playDiceGame () {
        int tmp;

        for (int i = 0; i < diceNum; i++) {
            // 주사위 객체 생성
              diceArr[i] = new Dice();
            // 주사위를 굴려야함
              // 합산
              tmp = diceArr[i].rollDice();
            System.out.printf("tmp = %d\n", tmp);
            sum += tmp;
        }

        checkWin(sum);
    }

    @Override
3)  public String toString() {
        return "DiceManager{" +
                "diceArr=" + Arrays.toString(diceArr) +
                ", sum=" + sum +
                '}';
    }
}
```

생성자 (2)입력하면서 diceNum이 2가 됨.
diceArr의 크기가 된다. 근데.. New Dice[2]는 무슨뜻일까??
DICE객체를 배열 2개에 각각 넣겠다는 뜻 . 뒷장에서 설명하도록 하겠음

오. 배열 형 Dice 객체 값의 이름이 diceArr이였구나.
그래서 diceArr[i].rollDice()매소드를 이렇게 호출할 수 있었구나.
그리고 dice형 객체를 배열로 만들었기 때문에,
배열 [0] = 객체(1) // 배열[1] = 객체(2) → 객체가 따로 두개가 있는거야.
그래서 값을 두개를 도출 해 낼 수 있다.

게임을 시작하겠다.
사용자 입력값만큼 FOR돌면서
diceArr[i]값에 new Dice()를 한다. 이때 new Dice()는 생성자다!!!!
Dice 클래스의 내용을 좀 봐야할 것 같은데.

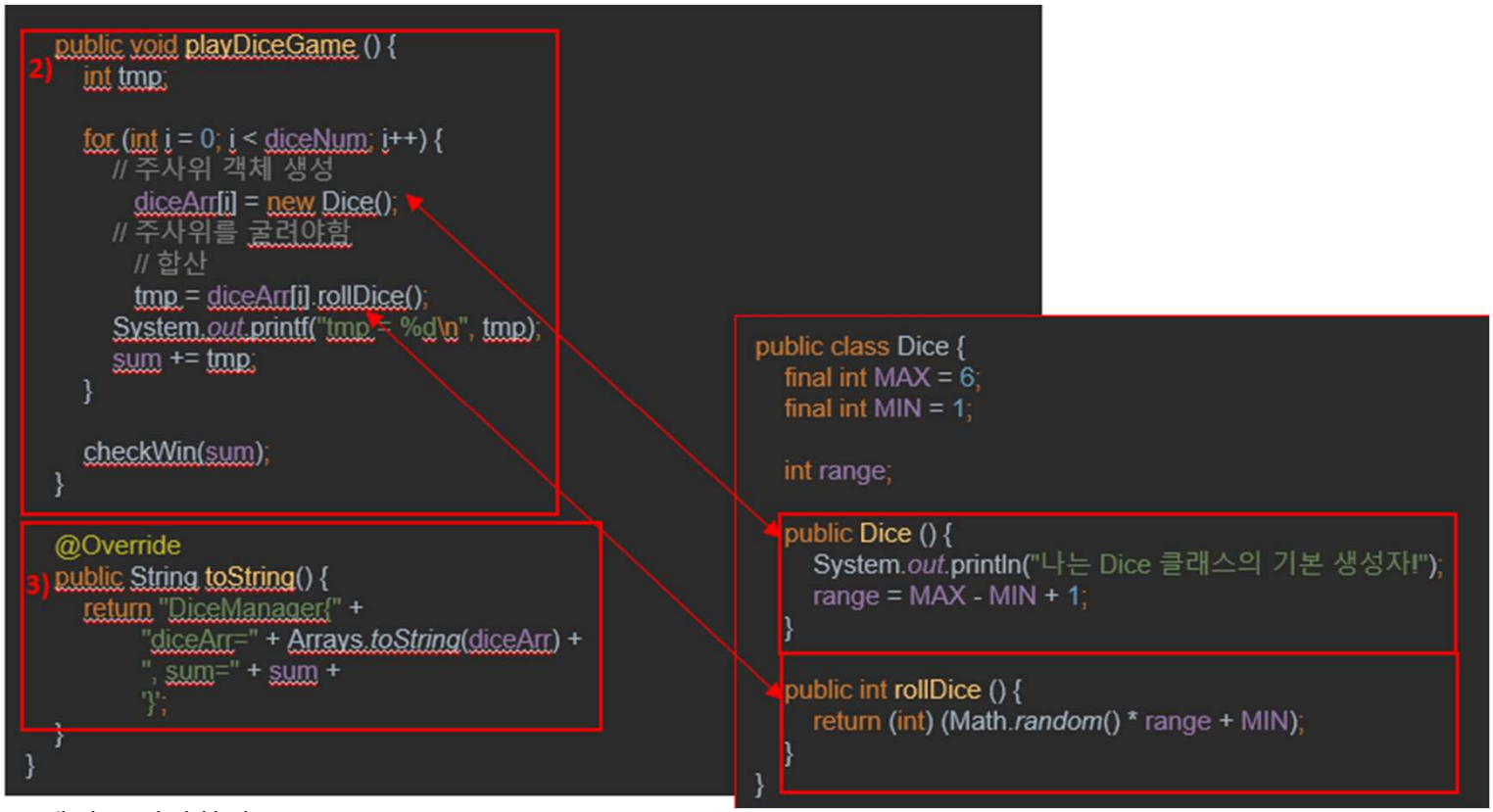
```
public class Dice {
    final int MAX = 6;
    final int MIN = 1;

    int range;

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

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

위에 부분 좀 더 자세히 보기.



게임을 시작하자.
For문으로 diceNum – 메인클래스에서 입력한 값만큼 반복한다.

diceArr[i] = new Dice(); → dice 생성자 불러온다.
Dice 생성자에는 랜덤의 범위 값을 가지고있다.

Tmp → diceArr[0] 객체일때 rollDice 매소드 사용 → 랜덤 값 도출
diceArr[1]일때 rollDice 매소드 사용 →랜덤값 도출
(이건 다음페이지에서 좀 더 얘기해보자)

그 값을 checkWin으로 간다. 이겼는지 안이겼는지 확인하는 매소드

```
public void checkWin (int sum) {
    if (sum % 2 == 0) {
        System.out.println("당첨입니다!!!");
    } else {
        System.out.println("올땐 쉬웠지 갈땐 손모가지닷!");
    }
}
```

마지막으로 toString 값 도출
diceArr 는 각 두개의 객체의 값이 있기 때문에, 두개 값이 나오지 않을까?
Sum은 sum한 값이 나오겠지? 결과값을 보자.

```
@Override
public String toString() {
    return "DiceManager{" +
            "diceArr=" + Arrays.toString(diceArr) +
            ", sum=" + sum +
            '}';
}
```

```java
Public class DiceGameTest {
    public static void main(String[] args) {
        // 주사위 2개를 굴려서 눈금의 합을 출력해봅시다.
        // 주사위의 개수 자체가 확장이 가능함
          DiceManager dm = new DiceManager(2);

        dm.playDiceGame();
        System.out.println(dm);
    }
}
```

```
나는 Dice 클래스의 기본 생성자!
tmp = 1
나는 Dice 클래스의 기본 생성자!
tmp = 5
당첨입니다!!!
DiceManager{diceArr=[Dice@12a3a380, Dice@29453f44], sum=6}

Process finished with exit code 0
```

결론 → DiceManager 생성자 2 호출하여 dice객체의 배열 길이 지정
→ playDiceGame 매소드 호출해서 각 배열에 값 입력 +sum
→ sum값으로 if문으로 결과 값 도출
→ toString으로 값 도출

```java
public class DiceManager {
    int diceNum;

    Dice[] diceArr;
    int sum;

public DiceManager (int diceNum) {
    sum = 0;
    this.diceNum = diceNum;

    diceArr = new Dice[diceNum];
}

public void playDiceGame () {
    int tmp;

    for (int i = 0; i < diceNum; i++) {
        // 주사위 객체 생성
          diceArr[i] = new Dice();
        // 주사위를 굴려야함
          // 합산
          tmp = diceArr[i].rollDice();
        System.out.printf("tmp = %d\n", tmp);
        sum += tmp;
    }

    checkWin(sum);
}
```

Dice[] diceArr ➔ 배열에 int형 배열이 있듯이 객체형 배열도 있는것이다.
Dice 객체의 배열이라고 들어봤니? 이게 바로 그거란다.
처음엔 객체를 만들기만 했음.

그 객체를 생성하는건 바로 diceArr = new Dice[diceNum]에서다!!
이게 무슨뜻이냐면
diceNum 개수만큼의 dice객체를 만들어라! → 총 객체를 2개 만든다
diceArr[0] → dice객체1 / diceArr[1] → dice객체2 ➔ 각각 다른 랜덤 값을 가짐
왜냐면 dice는 retur값이 있는데 랜덤 값이 바로 호출된다..!

diceArr[i] = new Dice(); 는 바로 생성자이다!
바로 위에 객체를 호출했다면 이번에는 생성자를 호출한것...!!

이 차이를 잘 알자!

메모리 영역

```
-----------------------------------
| Dice 객체 1 주소 | Dice 객체 2 주소 |    diceArr
-----------------------------------
       [0]                [1]

  각각의 Dice 객체는 아래와 같음
    -------------     -------------
    | MAX       |     | MAX       |
    | MIN       |     | MIN       |
    | range     |     | range     |
    -------------|     -------------|
    | Dice()    |     | Dice()    |
    | rollDice()|     | rollDice()|
    -------------     -------------

   rollDice를 하고 return을 하면
   배열 객체에 각각의 Dice 객체를
   가지고 있는 것이 의미가 없다고 한 이유
   만약 Dice 객체에 각 주사위가 뽑은 숫자값이 기록된다면
   각각의 주사위 객체를 유지하는 것이 의미가 있음
```

진짜 마지막....

```java
@Override
public String toString() {
    return "DiceManager{" +
            "diceArr=" + Arrays.toString(diceArr) +
            ", sum=" + sum +
            '}';
}
```

toString을 출력했는데
값이 이렇게 나왔네?
즉 diceArr 배열의 주소값을 도출해낸것.

각 객체의 주소값이 다르다는 것을 참고하길 바란다.

```
나는 Dice 클래스의 기본 생성자!
tmp = 1
나는 Dice 클래스의 기본 생성자!
tmp = 5
당첨입니다!!!
DiceManager{diceArr=[Dice@12a3a380, Dice@29453f44], sum=6}

Process finished with exit code 0
```

학생 질문 :
***주소값이 지정되면 주소값으로 불러올수도 있는것아닌가요?
자바는 지원을 안한다 (주소값은 가상메모리임 / 직접접근이 안됨)

선생님 왈 :
rollDice를 하고 return을 하면
배열 객체에 각각의 Dice 객체를
가지고 있는 것이 의미가 없다고 한 이유
만약 Dice 객체에 각 주사위가 뽑은 숫자값이 기록된다면
각각의 주사위 객체를 유지하는 것이 의미가 있음

```java
public class Dice {
    final int MAX = 6;
    final int MIN = 1;

    int range;

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

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

선생님의 뜻은.
아예 값이 dice객체 안에 출력되어 있었다면
각각의 객체 배열을 만들때 값은 동일하게 되기 때문에
굳이 배열을 만들 필요가 없다는 뜻인것같다.

질문3. (1)난수(4~97) // (2)난수(65~ 90 or 97~ 122) //
(3)65~122 사이중에 난수 (65~ 90 or 97~ 122)

```
public class RandomGeneratorTest {
    public static void main(String[] args) {
        RandomGenerator rg = new RandomGenerator(4, 97);

        if (!rg.confirmRandom()) {
            System.out.println("난수 생성에 문제가 있음");
        } else {
            System.out.println("난수 생성: " + rg.intGenerate());
        }

        RandomGenerator rg2 = new RandomGenerator(
                65, 90, 97, 122
        );

        if (!rg2.confirmComplicatedRandom()) {
            System.out.println("난수 생성에 문제가 있음");
        } else {
            System.out.println("복합 난수 생성: " + rg2.complicatedRandom());
        }

        RandomGenerator rg3 = new RandomGenerator(
                65, 122,
                65, 90, 97, 122
        );

        System.out.println("조건부 난수 생성: " + rg3.conditionRandom());
    }
}
```

복잡해보이지만, 하나씩 떼어서 생각해보면
셋중에 이게 제일 간단하다. 하나씩 보러가자.

```
RandomGenerator rg = new RandomGenerator(4, 97);

if (!rg.confirmRandom()) {
    System.out.println("난수 생성에 문제가 있음");
} else {
    System.out.println("난수 생성: " + rg.intGenerate());
}
```

1) RandomGeneration생성자에 4,97 입력해서,
들어가면 그 값이 최대값 최소값을 입력.

2) If confirmRandom이 false이라면 → 난수 생성에 문제있음

3) else, 즉 true면 intGenerate 매소드 출력!

```
public class RandomGenerator {
    int intMax, intMin, intRange;
    final int CHECK_NUM = 100000;


    public RandomGenerator (final int intMin, final int intMax) {
        this.intMin = intMin;
        this.intMax = intMax;

        intRange = intMax - intMin + 1;
    }
```

```
public boolean confirmRandom () {
    int tmp;

    for (int i = 0; i < CHECK_NUM; i++) {
        tmp = intGenerate();

        if (tmp < intMin || tmp > intMax) {
            return false;
        }
    }

    return true;
}
```

이 매소드는 boolean형이다, 출력은 true아님 false
For문 통해 10000 번 반복(check_num)하면서
intGenerate 매소드를 호출
해당 매소드는 랜덤 int값을 retur한다.!
만약 그 값이 intMin 보다 적고 intMax보다 많으면 false
아니면 true로 출력하여 tmp값을 출력하게 도와준다.

```
public int intGenerate () {
    return (int) (Math.random() * intRange +
intMin);
}
```

```java
public class RandomGeneratorTest {
    public static void main(String[] args) {

        RandomGenerator rg2 = new RandomGenerator(
                65, 90, 97, 122
        );

        if (!rg2.confirmComplicatedRandom()) {
            System.out.println("난수 생성에 문제가 있음");
        } else {
            System.out.println("복합 난수 생성: " + rg2.complicatedRandom());
        }
```

첫번째, RandomGenerator의 int숫자 4개가 있는 생성자를 만든다.
만들어서 보면, 각 값에 max, min을 각 두개씩 만든다.
Range도 두개 만든다. 랜덤 범위가 두곳이기 때문

```java
public class RandomGenerator {
    int intMax, intMin, intRange;
    int intMax2, intMin2, intRange2;

    final int TWO = 2;
    final int CHECK_NUM = 100000;

    public RandomGenerator (
            final int intMin, final int intMax,
            final int intMin2, final int intMax2) {

        this.intMin = intMin;
        this.intMax = intMax;

        this.intMin2 = intMin2;
        this.intMax2 = intMax2;

        intRange = intMax - intMin + 1;
        intRange2 = intMax2 - intMin2 + 1;
    }
```

두번째, confirmComplicatedRandom에서 true면 복합 난수 값이 생성되도록 한다
그 랜덤 문 안으로 들어가보면, 10000번동안 tmp를한다?
intGenarate를 보자 랜덤값 도출 // 그 랜덤값이 65~90 해당되지 않으면 false
그리고 tmp2 도 랜덤값 도출 // 그 랜덤 값이 97~122 해당되지 않으면 false
나머지는 true로 끝내기

```java
public boolean confirmComplicatedRandom () {
    int tmp, tmp2;

    for (int i = 0; i < CHECK_NUM; i++) {
        tmp = intGenerate();

        if (tmp < intMin || tmp > intMax) {
            return false;
        }

        tmp2 = intGenerate2();

        if (tmp2 < intMin2 || tmp2 > intMax2) {
            return false;
        }
    }

    return true;
}
```

```java
public int intGenerate () {
    return (int) (Math.random() * intRange + intMin);
}
public int intGenerate2 () {
    return (int) (Math.random() * intRange2 + intMin2);
}
```

```java
public int complicatedRandom () {
    if (percent50() == 0) {
        return intGenerate();
    } else {
        return intGenerate2();
    }
}
```

세번째, true면 complicateRandom생성.
확률은 반반 / 랜덤으로 0아니면 1 도출하게 한다.
만약 0이면 65~90 사이 랜덤값 도출
만약 1이면 979~122사이 랜덤값 도출

```java
public int percent50 () {
    return (int) (Math.random() * TWO);
}
```

마지막..

```
public class RandomGeneratorTest {
    public static void main(String[] args) {

        System.out.println("조건부 난수 생성: " + rg3.conditionRandom());
    }
}
```

conditionRandom 매소드로 가자.

++CNT라서 시작부터 1번째 가능
Rand값으로 랜덤값 도출
만약 그 rand값이 65~ 90 or 97~ 122 가 아니라면 false
맞다면 true를 하면서 rand를 return한다.

```
public int conditionRandom () {
    int rand, cnt = 0;

    do {
        System.out.printf("%d 번째\n", ++cnt);
        rand = intGenerate();
    } while (isRandomNotOk(rand));

    return rand;
}
```

```
public boolean isRandomNotOk (int rand) {
    if ((rand >= condMin && rand <= condMax) ||
            (rand >= condMin2 && rand <= condMax2)) {
        return false;
    }

    return true;
}
```

```
public int intGenerate () {
    return (int) (Math.random() * intRange + intMin);
}
```

```
난수 생성: 78
복합 난수 생성: 67
1 번째
조건부 난수 생성: 80
```