2021.05.27 Java

```
class StudyStatic{

    public final int _FINAL = 5;
    public static int _static = 5;
```

〈Final 상수와 Static Data의 차이점〉

final은 수정 불가 / static은 수정 가능

〈class와 배열의 단순비교〉

배열은 같은 type의 data들을 묶을 수 있지만 다른 type의 data는 불가능.

class는 다른 type도 묶을 수 있음.

```java
public void if_Dice(int curDice1, int curDice2,
                    int targetDice1, int targetDice2) {
    curDice2 = 0;

    if (curDice1 % 2 == 0) {
        curDice2 = getRanDice( range: 6,  offset: 1);
    }
    switch (curDice2) {
        case 1:
            curDice1 += curDice1 + 3;
            break;
        case 3:
            targetDice1 += targetDice1 - 2;
            break;
        case 4:
            curDice1 = 0;
            break;
        case 6:
            curDice1 += (curDice1 + curDice2) * 2;
            targetDice1 += (targetDice1 + targetDice2) * 2;
            break;
        default:
            break;
```

《객체 전달 / 값 전달의 주요한 차이점 》

《 코드 전체는 20210526 Review&Question pdf 참조 》

왼쪽과 같은 경우는 참조를 전달할 방법이 없기 때문에 변수 자체를 수정하는게 불가능.
((C/C++에서는 가능한 방식 .. 포인터 연산방식이기 때문에))

curDice1 / targetDice1 / curDice2 / targetDice2 는 객체가 아니다
= 메모리 할당이 되어있지 않다.

```java
if_Dice(user_Dice, user_Dice2, computer_Dice, computer_Dice2);
```

이런 식으로 매개변수를 넣었을 때.
user_Dice = 3인 경우.
curdice1이 받는 것은 user_Dice라는 '객체'가 아니라 3이라는 '값'이다.
즉 왼쪽에 있는 code에서 컴퓨터가 이해하기를
case 1: 3 += 3 + 3 // 컴퓨터: ....???????
이런식으로 받아들인다.

이럴 때 해결하는 방법 중 하나는 》》

```java
private void checkSkill (int[] curDice, int[] targetDice) {
    switch (curDice[SECOND_IDX]) {
        case 1:
            curDice[TOTAL_IDX] = curDice[FIRST_IDX] + 3;
            break;
        case 3:
            targetDice[TOTAL_IDX] = targetDice[FIRST_IDX] - 2;
            break;
        case 4:
            curDice[TOTAL_IDX] = 0;
            break;
        case 6:
            curDice[TOTAL_IDX] = (curDice[FIRST_IDX] + curDice[SECOND_IDX]) * 2;
            targetDice[TOTAL_IDX] = (targetDice[FIRST_IDX] + targetDice[SECOND_IDX]) * 2;
            break;
        default:
            curDice[TOTAL_IDX] = curDice[FIRST_IDX] + curDice[SECOND_IDX];
            break;
    }
}
```

```java
int[] comDice;
int[] usrDice;
```

```java
private void checkMagicDiceSkill () {
    // 사용자 관점에서의 2번째 주사위 스킬 발동
    checkSkill(usrDice, comDice);
    // 컴퓨터 관점에서의 2번째 주사위 스킬 발동
    checkSkill(comDice, usrDice);
```

《배열을 사용하면 된다》

**** 중요 ****

배열 객체는 결국 메모리고

배열이름(배열명)은 이 배열 객체의 대표로

전체 메모리를 전달하게 된다.

int[] arr에서 arr은 배열 객체 전체

배열의 이름은 배열의 대표.

이는 결국 객체가 전달된다는 의미 › 원본 형태로 전달

but. 배열 전달시에

arr[2] 와 같은 특정 index의 값을 지정하는

경우에는 변수와 동일하게 '값'을 복사해서

전달한다.

class를 메모리에 올린 객체를 전달하면

말 그대로 객체가 전달.

나머지는 전부 값이 복사되어 전달된다.

원리 관점에서 얘기하자면

메모리를 바라보는지(point) 》 '객체'

메모리 내부에 들어있는 값을 보는지(point) 》 '값'

// 객체(메모리)를 요청하면 원본을 준다.

// 객체내에 들어있는 '객체'를 요청하면 원본을 줌 《《주의》》

// 객체내에 들어있는 '값'을 요청하면 이 값을 복사해서 줌

// 인간 vs 복제 인간

// 둘은 같은 인간일까요 ? 다른 인간일까요 ?

// DNA 정보는 같지만 나는 나. 너는 너 관점이라고 보면 되겠다

// 게터에서도 값을 리턴하는 것은 전부 복제인간이다.

// 게터에서도 객체를 리턴하는 것은 메모리 즉. 원본이다. 《 객체내에 들어있는 '객체'를 요청하면 원본을 줌 》》

// 메모리의 개념이 헷갈린다면.

// 배열명으로 전달하면 - 객체 전달(메모리 전달)

// 배열의 index를 전달하면 - 값이 복사되어 전달

// class의 객체를 전달하면 - 객체 전달(메모리 전달)

// 그 외의 나머지(변수)를 전달한다 - 값이 복사되어 전달

```java
class CloneMemory {
    int[] arr;
    int num;

    public CloneMemory () {
        arr = new int[3];
        num = 3;
        for (int i = 0; i < 3; i++) {
            arr[i] = (int)(Math.random() * 6 + 1);
        }
    }
    public void reRandArr () {
        for (int i = 0; i < 3; i++) {
            arr[i] = (int)(Math.random() * 6 + 1);
        }
    }
    public void reNum () { num = 7; }
    //...
    public int[] getCloneArr () { return arr; }
    public int getCloneVariable () { return num; }
    public String toString () {
        return "arr[0] = " + arr[0] +
               ", arr[1] = " + arr[1] +
               ", arr[2] = " + arr[2];
    }
```

〈객체 전달과 값 전달의 주요한 차이점 예제〉

class에서 arr배열과 정수 num을 setting.

- arr[]는 index 3개에 랜덤한 정수 저장
- num = 3

main에서 위의 값들을 호출해서
임의의 다른 배열/변수에 저장.

그리고
reRandArr / ReNum을 통해서 arr[]와 num의 값을 재설정한 후
다시 호출 했을 때
임의의 배열/변수에 저장했던 값들이
변하는가. 변하지 않는가로
객체가 전달되었는지 값이 전달되었는지 판단 할 수 있다.

```java
public class _2nd_CloneConceptEnhance {
    public static void main(String[] args) {
        CloneMemory cm = new CloneMemory();
        System.out.println(cm);

        int[] save = cm.getCloneArr();

        System.out.printf("save[0] = %d, save[1] = %d, save[2] = %d\n",
                save[0], save[1], save[2]);

        cm.reRandArr(); // arr[]의 값들 재설정

        System.out.println("객체에 접근해 출력");
        System.out.println(cm);

        System.out.println("save[]에 저장해두었던 정보 출력");
        System.out.printf("save[0] = %d, save[1] = %d, save[2] = %d\n",
                save[0], save[1], save[2]);

        int num = cm.getCloneVariable();

        System.out.println("객체 내 변수값 획득: " + num);

        cm.reNum(); // class data인 num의 값 재설정

        System.out.println("변경 후 사전 획득한 정보 재출력: " + num);
        System.out.println("변경 정보 파악: " + cm.getCloneVariable());
```

toString으로 이런식으로도 값 불러오기 가능

```
arr[0] = 3, arr[1] = 5, arr[2] = 5
save[0] = 3, save[1] = 5, save[2] = 5
객체에 접근해 출력
arr[0] = 1, arr[1] = 4, arr[2] = 3
save[]에 저장해두었던 정보 출력
save[0] = 1, save[1] = 4, save[2] = 3
객체 내 변수값 획득: 3
변경 후 사전 획득한 정보 재출력: 3
변경 정보 파악: 7
```

처음 arr[]에 설정된 값은

arr[0] = 3, arr[1] = 5, arr[2] = 5

그 값을 main의 save[]에 저장하고

다시 reRandArr로 arr[]의 값들 재설정.

재설정된 값 〉 arr[0] = 1, arr[1] = 4, arr[2] = 3

그리고 전에 설정해두었던 save[]의 값들을 다시 출력해보면

재설정된 arr[]의 값과 동일하게

save[0] = 1, save[1] = 4, save[2] = 3로 바뀜.

즉, 값이 아니라 객체를 전달했기 때문에

reRandArr로 arr[]를 재설정하면 save[]도

같은 값으로 재설정되는 것을 볼 수 있음.

```java
public class _2nd_CloneConceptEnhance {
    public static void main(String[] args) {
        CloneMemory cm = new CloneMemory();
        System.out.println(cm);

        int[] save = cm.getCloneArr();

        System.out.printf("save[0] = %d, save[1] = %d, save[2] = %d\n",
                save[0], save[1], save[2]);

        cm.reRandArr(); // arr[]의 값들 재설정

        System.out.println("객체에 접근해 출력");
        System.out.println(cm);

        System.out.println("save[]에 저장해두었던 정보 출력");
        System.out.printf("save[0] = %d, save[1] = %d, save[2] = %d\n",
                save[0], save[1], save[2]);

        int num = cm.getCloneVariable();

        System.out.println("객체 내 변수값 획득: " + num);

        cm.reNum(); // class data인 num의 값 재설정

        System.out.println("변경 후 사전 획득한 정보 재출력: " + num);
        System.out.println("변경 정보 파악: " + cm.getCloneVariable());
```

toString으로 이런식으로도 값 불러오기 가능

```
arr[0] = 3, arr[1] = 5, arr[2] = 5
save[0] = 3, save[1] = 5, save[2] = 5
객체에 접근해 출력
arr[0] = 1, arr[1] = 4, arr[2] = 3
save[]에 저장해두었던 정보 출력
save[0] = 1, save[1] = 4, save[2] = 3
객체 내 변수값 획득: 3
변경 후 사전 획득한 정보 재출력: 3
변경 정보 파악: 7
```

반대로 num의 경우에는.

처음에 설정한 num = 3을

main의 변수 num(class의 num과는 다른 num)의 값으로 저장하고

다시 class에서 num = 7로 재설정.

그리고 main에서 그 값들을 출력해보면

재설정 하기 전에 저장했던 값은 그대로 3인 것을 볼 수 있다.

즉. 변수의 값을 전달하고 저장한 것이기 때문에 그 값은 그대로.

```java
// 궁금한점.
// 과연 그럼 save[]안에 저장된 값들을 재설정하면
// class 내부의 arr[]도 같은 값으로 변할까????
for (int i = 0; i < 3; i++) {
    save[i] = (int)(Math.random() * 6 + 1);
}
System.out.println("save[]를 재설정하고 arr[]도 변하는지 볼 것");
System.out.printf("save[0] = %d, save[1] = %d, save[2] = %d\n",
        save[0], save[1], save[2]);

System.out.println("arr[]도 변했을까");
System.out.println(cm);
//
```

그럼 갑자기 궁금한점..

과연 main에서 save[]의 값을 재설정하면

class에 있는 arr[]의 값들도 재설정된 save[]와 같은 값으로 변할 것인가?

객체가 전달되고 있는 것이기 때문에 이론상 당연히 그래야 한다.

결과 또한 이론에 맞게

save[]의 값을 재설정하자 class의 arr[]의 값들도 그와 같은 값으로 재설정 됐다.

```
arr[0] = 3, arr[1] = 1, arr[2] = 5
save[0] = 3, save[1] = 1, save[2] = 5
객체에 접근해 출력
arr[0] = 3, arr[1] = 5, arr[2] = 1
save[]에 저장해두었던 정보 출력
save[0] = 3, save[1] = 5, save[2] = 1
save[]를 재설정하고 arr[]도 변하는지 볼 것
save[0] = 2, save[1] = 4, save[2] = 5
arr[]도 변했을까
arr[0] = 2, arr[1] = 4, arr[2] = 5
객체 내 변수값 획득: 3
변경 후 사전 획득한 정보 재출력: 3
변경 정보 파악: 7
```

〈toString〉

```
//    public String toString () {
//        return "arr[0] = " + arr[0] +
//                ", arr[1] = " + arr[1] +
//                ", arr[2] = " + arr[2];
//    }
}
public class _2nd_CloneConceptEnhance {
    public static void main(String[] args) {
        CloneMemory cm = new CloneMemory();
        System.out.println(cm);

        int[] save = cm.getCloneArr();
```

```
_2nd_CloneConceptEnhance
CloneMemory@4eec7777
save[0] = 1, save[1] = 6, save[2] = 1
객체에 접근해 출력
CloneMemory@4eec7777
save[]에 저장해두었던 정보 출력
save[0] = 2, save[1] = 2, save[2] = 5
save[]를 재설정하고 arr[]도 변하는지 볼 것
save[0] = 5, save[1] = 4, save[2] = 1
arr[]도 변했을까
CloneMemory@4eec7777
객체 내 변수값 획득: 3
변경 후 사전 획득한 정보 재출력: 3
변경 정보 파악: 7
```

toString은 무엇인가..?

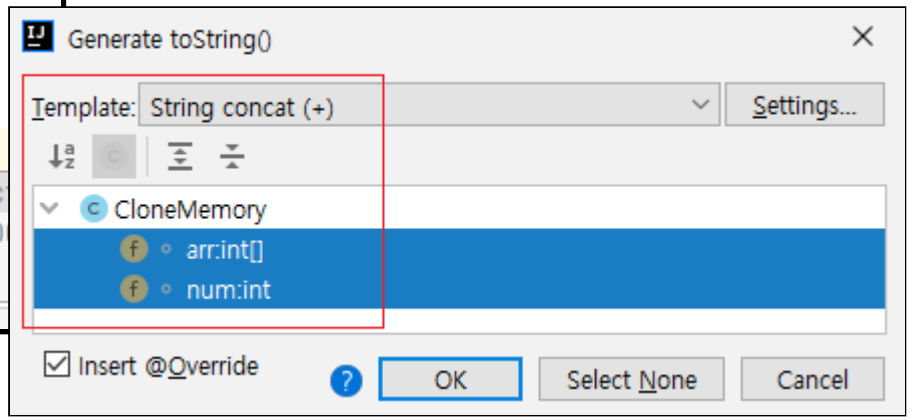
toString을 없애고

instance cm을 printout 하면

이런 주소값??들이 나온다.

```
//                ", arr[2] = " + arr[2];
//    }

    to
    public String toString() {...}                    Objec
    todo                                          adds // TOD
Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards  Next Tip
```



편의기능을 이용해서 toString을 다시 호출하면

```java
@Override
public String toString() {
    return "CloneMemory{" +
            "arr=" + Arrays.toString(arr) +
            ", num=" + num +
            '}';
}
```

〈to String으로는 자동완성 가능.

객체의 정보 출력에 사용된다.

'인터페이스'를 배울 때 더 자세하게 배울 예정〉

이런 method가 호출된다.

```
_2nd_CloneConceptEnhance
"C:\Program Files\Java\jdk-16\bin\java.
CloneMemory{arr=[3, 6, 5], num=3}
save[0] = 3, save[1] = 6, save[2] = 5
객체에 접근해 출력
CloneMemory{arr=[6, 5, 6], num=3}
save[]에 저장해두었던 정보 출력
save[0] = 6, save[1] = 5, save[2] = 6
save[]를 재설정하고 arr[]도 변하는지 볼 것
save[0] = 4, save[1] = 2, save[2] = 1
arr[]도 변했을까
CloneMemory{arr=[4, 2, 1], num=3}
객체 내 변수값 획득: 3
변경 후 사전 획득한 정보 재출력: 3
변경 정보 파악: 7
```

그리고 그 method안에 지정된 형식대로

print가 된다.

toString은 instance를 생성했을 때

그 안에 지정된 형식대로

간편하게 print할 수 있게 해주는 것인가....??

```java
import java.util.ArrayList;
public class _3rd_ArrayList {
    public static void main(String[] args) {
        //...

        ArrayList<String> lists = new ArrayList<String>();
        lists.add("빵");
        lists.add("버터");
        lists.add("우유");
        lists.add("계란");
        lists.add("쥬스");
        lists.add("베이컨");
        lists.add("파스타");
        lists.add("비프샐러드");
        lists.add("피자");

        for(String list : lists){
            System.out.println("현재 항목은: "+ list);
        }
```

```
"C:\Program Files\Java
현재 항목은: 빵
현재 항목은: 버터
현재 항목은: 우유
현재 항목은: 계란
현재 항목은: 쥬스
현재 항목은: 베이컨
현재 항목은: 파스타
현재 항목은: 비프샐러드
현재 항목은: 피자
```

〈Array List - 동적 배열〉

일반적인 배열은 배열의 size를 정하고 사용한다.
그 size는 변경 할 수 없고 그렇다고 그 배열의 크기를 미리 크게 만들어 놓는다면
반복문에서 error가 생길 확률도 높고 그에 따른 조건도 만들어줘야 한다.
그리고 일반적인 배열은 원소의 삭제가 불가능하다.

이러한 단점들을 보완할 수 있는 것이 Array List - 동적배열.
Array List도 heap을 이용한 동적할당을 수행하며
그 크기에대한 제한이 없다.
그리고 원소를 추가/수정/삭제 할 수 있다.

- how to use

ArrayList<data type> 배열이름 = new ArrayList<data type>()

```
// 일반 배열과의 차이점
// 배열은 메모리가 "연속적"으로 배치. ArrayList는 "불연속 배치"
// ArrayList
// | 데이터 | 다음링크 | ---> | 데이터2 | 다음링크 | ---> |데이터3 | 다음링크 | ---> ...
// 속도느림 // but 유연성 극대화
// 몇 개가 들어올지 모른다? >> arraylist

// 일반 배열
// | 데이터1 | 데이터2 | 데이터3 | 데이터4 | 데이터5 | 데이터6 | .....
// 속도빠름 / but index값이 정해져있음(개수가 정해져있음)
```

```java
import java.util.ArrayList;
import java.util.Scanner;
class Shop {
    ArrayList<String> lists;
    Scanner scan;

    public Shop () {
        lists = new ArrayList<String>();
        scan = new Scanner(System.in);
    }

    public void order () {
        System.out.print("필요한 물품을 말씀하세요: ");
        lists.add(scan.nextLine());
    }
    public void cancelOrder () {
        System.out.print("취소할 물품을 말씀하세요: ");
        lists.remove(scan.nextLine());
    }
    //...
    @Override
    public String toString() {
        return "Shop{" +
                "lists=" + lists +
                '}';
    }
```

```java
public class _3rd_ShopArrayTest {
    public static void main(String[] args) {
        Shop shopping = new Shop();

        for(int i = 0; i < 3; i++) {
            shopping.order();
        }

        shopping.cancelOrder();
        System.out.println(shopping);
    }
```

```
"C:\Program Files\Java\jdk-16
필요한 물품을 말씀하세요: 컴퓨터
필요한 물품을 말씀하세요: 카메라
필요한 물품을 말씀하세요: 가방
취소할 물품을 말씀하세요: 가방
Shop{lists=[컴퓨터, 카메라]}
```

(ex)임의의 배열 arr1)

원소 추가) arr1.add("add")

원소 삭제) arr1.remove("remove") 또는 arr1.remove(2) {(2)는 두번째니까 index[1]이 삭제}

원하는 위치의 원소 가져오기) arr1.get(2) {같은 원리로 index[1]이 호출}

원하는 위치에 원소 추가) arr1.add(0, "원소추가") index[0]에 "원소추가" 추가

나머지 원소들은 뒤로 밀림

원하는 위치의 원소 값 수정) arr1.set(0, "수정할 값") index[0]의 값이 "수정할 값"으로 수정

특정 값의 존재여부 판단) arr1.contains("존재하는지 궁금한 값")

arr1[]의 원소 중에 "존재하는지 궁금한 값"이 존재하는지 판단 후 true/false로 반환.

특정 값의 위치 판단) arr1.indexOf("위치 알고싶은 원소")

"위치 알고싶은 원소"가 index[2]에 있다면 2를 반환

원소의 존재 여부 판단) arr1.isEmpty() - 배열이 원소를 가지고 있으면 true / 아니면 false

모든 원소 삭제) arr1.clear()

```java
import java.util.ArrayList;
public class _4th_QuIz48 {
    public static void main(String[] args) {
//        뽑기 게임
//        중복이 발생하지 않게 랜덤한 배열에 사람 이름을 무작위로 믹스한다.
//                그리고 당첨자 번호를 3개 뽑도록 한다(마찬가지로 중복 x)
//        당첨된 사람의 이름을 출력.
        String[] studentlist = {"박세진", "김창욱", "김민규", "김중연", "문성호",
                "강병화", "최승현", "유종현", "한상우", "전승리",
                "이경환", "최준환", "김원석", "여인준", "이태양",
                "김윤영", "정도영", "황정아", "임초롱", "김남교",
                "이주형", "김도연", "최혜주", "김도혜", "고재권",
                "임익환", "안보미", "이상훈"};
        ArrayList<String> lotto = new ArrayList<String>();

        for (int i = 0; i < 3; i++) {
            int num = (int) (Math.random() * 28);  // 중복이 될 수 있음
            lotto.add(studentlist[num]);
        }
        for (String lottowinner : lotto) {
            System.out.println("당첨자는: " + lottowinner);
        }
```

```
_3rd_ShopArrayTest
"C:\Program File
당첨자는: 유종현
당첨자는: 김중연
당첨자는: 김중연
```

〈Quiz. 48 - 수업 중 내 풀이〉

수업 중 내 풀이는

Math.random에서 중복이 나올 수 있기 때문에

실패.

〈Quiz. 48 -강사님 풀이 듣고 pdf 다시 작성〉