요약 정리_진유진

회원 클래스 구현

```
// 회원 이메일 주소 입력
// 회원 비밀번호 입력
// 올바르게 들어갔는지 확인 출력

// 회원 클래스 생성
class Member {

    // 회원 이메일 주소 변수 생성
    private final String email;
    // 회원 비밀번호 변수 생성
    private final String password;

    // 생성자 매개변수 추가
    // 아직 이 부분이 필요한 정확한 이유는 모르겠음
    Member(String email, String password) {
        this.email = email;
        this.password = password;
    }

    // toString()을 이용한 출력 형식
    // toString(): 객체가 가지고 있는 정보나 값들을 문자열로 만들어서 리턴하는
메소드
    @Override
    public String toString() {
        return "Member{" +
                "email='" + email + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
}

public class Summary {
    public static void main(String[] args) {
        // new 연산자 이용하여
        // Member 클래스 타입의 member 변수 생성
        // 따라서 Member 클래스의 메소드 toString()을 통해 출력
```

```
       Member member = new Member("test@test.com", "test123");
        System.out.println(member);
    }
}
```

```
Member member = new Member();
```

"new 연산자를 이용하여
Member 클래스 타입의 member 변수 생성
따라서 Member 클래스의 메소드 toString()을 통해 출력"
했다고 설명하면 OK (편집됨)

Person 클래스 구현

```
// 사람 나이
// 사람 이름
// 확인 출력

// Person 클래스 생성
class Person {
    // 사람 나이 변수 생성
    private int age;
    // 사람 이름 변수 생성
    private String name;

    // 생성자 매개변수 추가
    Person(int age, String name) {
        this.age = age;
        this.name = name;
    }

    // toString()을 이용한 출력
    @Override
    public String toString() {
        return "Person{" +
                "age=" + age +
                ", name='" + name + '\'' +
                '}';
    }
}

public class Summary2 {
```

```
    public static void main(String[] args) {
        // new 연산자를 이용하여 Person 클래스의 메소드를 이용하여
        // 원하는 결과를 출력할 수 있는 person 변수 생성
        Person person = new Person(13, "김이름");
        System.out.println(person);
    }
}
```

커스텀랜덤

static 은 언제나 메모리에 상주하여 stack 도 heap 도 아니다
따라서 new 없이 사용이 가능하다
대표적으로 main, Math.random 같은 것들

```
public static int generateNumber (int min, int max)
{return(int)(Math.random() * (max - min) + 1)) + min;}
```

이렇게 min 과 max 사이를 랜덤으로 왔다갔다 하는 generateNumber 메소드를 생성 (편집됨)

CustomRandom 구현

```
// 주사위 값 등 min 과 max 만 있다면 언제든 이용 가능한 CustomRandom 을
만들어보자
// CustomRandom 은 heap 이나 stack 에 저장되지 않는 static

public class Summary4 {
    public static int forSummary3 (int min, int max) {
        return (int)(Math.random() * (max - min +1)) + min;
    }
}
```

주사위 문제 1 구현

```
// 주사위 2개 생성
// 2개의 주사위를 굴려 두 주사위의 합이 4의 배수가 된다면 승리
// 주사위 굴리기
// 주사위의 합 구하기
// 합이 4의 배수가 되는지 구하기
// 승패 결정
// 그 외의 케이스는 패배 처리하도록 만들기

import utility.random.CustomRandom;

public class Summary3 {
    public static void main(String[] args) {
        // 주사위 2개 생성
        // 주사위 굴리기
        final int DICE_MAX = 6;
        final int DICE_MIN = 1;
        final int DICE_COUNT_MAX = 2;
        int DiceSum= 0; // 초기화 해줘야 함
        final int DICE_WIN_DECISION = 4;
        int[] diceNumArr = new int[DICE_COUNT_MAX];

//        int Dice1 = (int)(Math.random() * (DICE_MAX - DICE_MIN +1)) +1;
//        int Dice2 = (int)(Math.random() * (DICE_MAX - DICE_MIN +1)) +1;;
        // 주사위 최대값과 최소값을 선언해 주어야겠다

        // 변수이름만 다르고 식이 같으니 for 문을 이용하여 만들어주자
//        for (int i = 0; i < DICE_COUNT_MAX; i++) {
//                            // 주사위 개수 최대값 생성
//            int DiceNumber = (int)(Math.random() * (DICE_MAX - DICE_MIN
+1)) + DICE_MIN;
//        }
        // 만들었으면 위의 똑같은 두 식은 주석처리 해주자

        // 또 여기서 우리가 더 편리하기 위해 배웠던 것: CustomRandom 적용
        for (int i = 0; i < DICE_COUNT_MAX; i++) {
            diceNumArr[i] = CustomRandom.generateNumber(DICE_MIN,
DICE_MAX);
            // summary4 이름으로 클래스에 만들었지만 저장하지 않아 원래 있던
            // CustomRandom 사용
```

```
            // 했으니 위의 같은 식은 주석 처리

            // 각 주사위의 결과 값 출력
            System.out.println((i + 1) + " 순서의 주사위 결과 = " +
diceNumArr[i]);
            //System.out.println("두 번째 주사위 결과  = " + diceNumArr[1]);

            // 주사위 합 구하기
            // 주사위 합 변수 생성
            DiceSum = DiceSum + diceNumArr[i];
            // 같은 식 DiceSum += diceNumberArray[i];
        }
        // 두 주사위의 합 출력 <<-- 실수
        System.out.println("결과 합 = " + DiceSum);

        // 합이 4의 배수가 되면 승리
        // 아니면 패배
        if (DiceSum % DICE_WIN_DECISION == 0) {
            System.out.println("승리!");
        } else {
            System.out.println("패배!");
        }
    }
}
// 이런 경우는 그냥 단순하게 하는 것이 더 나을 것!
// 루프 도는 것에 대한 이해가 더 필요할 것 같다
```

루프에 대한 이해

```
import utility.random.CustomRandom;

public class Summary5 {
    public static void main(String[] args) {
//        for(int i = 1; i <= 3; i ++) {
//            System.out.println("Hello");
//        }
        final int DICE_COUNT_MAX = 2;
        final int DICE_MAX = 6;
        final int DICE_MIN = 1;
        int[] diceNumArr = new int[DICE_COUNT_MAX];

        for (int i = 0; i < DICE_COUNT_MAX; i++) {
            diceNumArr[i] = CustomRandom.generateNumber(DICE_MIN,
DICE_MAX);
            System.out.println("Hi");
        } // Hi 가 두번 출력
    }
}
// 그런데 내가 출력하려했던 건 두 개
// System.out.println("첫 번째 주사위 결과 = " + diceNumArr[0]);
// System.out.println("두 번째 주사위 결과 = " + diceNumArr[1]);
// 즉 위 두문장이 두번씩 등장할 수 밖에 없었음
// 따라서 출력문은 하나로 두고 i 를 넣어 순서 두 개의 결과가 나오도록 했어야 함
// System.out.println("현재 주사위 순서: " + (i + 1) + ", 주사위 결과 = " +
diceNumArr[i]);
// 또한 결과 합까지 같은 루프에서 나오기 때문에 결과 합도 2 가지의 식이 나옴
// 따라서 결과는 루프 밖에서 나오도록 처리 했어야 함
// 루프 안에서 출력하면 조건문이 충족되는 만큼 수행문이 출력된다는 것 명심할 것!
```

주사위 문제 2 구현

```
// 주사위 4개 생성
// 2개의 주사위를 굴려 두 주사위의 합이 3 혹은 4의 배수가 된다면 승리
// 주사위 굴리기
// 주사위의 합 구하기
// 합이 3 혹은 4의 배수가 되는지 구하기
// 승패 결정
// 그 외의 케이스는 패배 처리하도록 만들기
// 클래스 없이 할 경우

import utility.random.CustomRandom;

public class Summary6 {
    public static void main(String[] args) {

        final int MAX_DICE_COUNT = 4;
        final int DICE_MAX = 6;
        final int DICE_MIN = 1;
        int[] diceArrayNumber = new int[MAX_DICE_COUNT];
        int DiceSum = 0;
        final int DICE_DECISION_WIN1 = 3;
        final int DICE_DECISION_WIN2 = 4;

        // 주사위 굴리기
        // 주사위 굴리기
        for (int i = 0; i < MAX_DICE_COUNT; i++) {
            diceArrayNumber[i] =
CustomRandom.generateNumber(DICE_MAX,DICE_MIN);
            // 이제 알맞는 변수들 생성해주기

            // 각 주사위 결과 출력
            System.out.println("i = " + (i+1) + " , 주사위 결과 = " +
diceArrayNumber[i]);

            // 주사위 합 구하고 출력
            DiceSum += diceArrayNumber[i];
            System.out.println("i = " + (i+1) +", 주사위 합은 = " +
DiceSum);
            }
            // 3이나 4의 배수면 승 아니면 패 출력
```

```
            if (DiceSum % DICE_DECISION_WIN1 == 0 || DiceSum %
DICE_DECISION_WIN2 == 0) {
                // 승패 가르는 변수 생성해주기
                System.out.println("승리!");
            } else {
                System.out.println("패배!");

        }
    }
}
// 맞긴 맞는데 썩 맘에 들지는 않음
// 클래스를 이용해서 구하는 방법 -> 연습
// 아직 안배운 문법도 있는 것 같음
// 진도 나가면서 꾸준히 반복해서 연습해볼 것
```

그 동안 배운 것들

변수만드는 방법

1. 데이터 타입을 적는다
- int 정수형
- float 실수형
- string 문자형
2. 변수 이름을 설정한다
- 변수 이름은 다른 사람들도 이해하기 쉽게 직관적으로 만드는 것을 추천
3. 필요하면 초기화를 진행한다

변수 테스트

```
package variable;

public class VariableTest {
    public static void main(String[] args) {
        System.out.println("variable(변수) 테스트");
```

- variable 이라는 패키지에 VariableTest 라는 클래스 생성
- public static void main(String[] args){}: 프로그램의 시작점
  -여기서 public 으로 인해 main 메서드가 다른 클래스에서 호출될 수 있음

```
        int appleCount = 3;
        int grapeCount = 5;
        int totalCount = appleCount + grapeCount;
```

```
        System.out.println("totalCount = " + totalCount);
```

- appleCount 를 정수형으로 선언하고 초기값으로 3 을 설정

```
        final float FULL_PERCENT = 100;
        final float TAX = 3.3f;
        int income = 1000000;

        System.out.println("프리랜서 세전 수입: " + income +
                ", 세후: " + income * (FULL_PERCENT - TAX) / FULL_PERCENT);
    }
}
```

- FULL_PERCENT 를 실수형으로 선언하고 final 을 통해 불변 상수로 만들어줌

---

final 을 사용하는 이유: 상수를 고정시켜 준다

상수를 고정시키는 것의 이점

1. 불편함 해소
예를 들어 final float TAX = 3.3f 를 고정시키지 않고 일일이 3.3f 를 썼다면 금리가 변경되었을 때 식이 간단하면 괜찮지만 그렇지 않은 경우 일일이 또 찾아서 변경해주어야 한다. 그러나 고정시키고 TAX 로 써준다면 그 값만 바꿔주면 자동으로 변경된다
2. 불변 객체
클래스를 인스턴스화 하여 객체를 만들었고 이것이 불변일 때 좋은 점
- 1.입금, 2.출금, 3.조회시스템을 생성한다고 할 때 이 변수들을 각각 final 로 고정시켜주지 않는다면 다른 사람이 코드를 건드렸을 때 1,2,3 을 잘못 설정하면 말 그대로 잘못된 결과가 출력되는 것
혹은 로직이 변경되어도 숫자로 대입했을 경우 일일히 바꿔주어야 하기 때문에 역시나 불편함 - 변경해주지 못하면 오류 발생따라서 final 로 고정을 시켜주어 잘못된 결과가 출력되는 것을 막아주어야 한다는 것

```
------------옳은 코드------------
public class BankingSystem {
    final int DEPOSIT = 1;
    final int WITHDRAW = 2;
    final int INQUIRY = 3;

    public void doOperation(int op) {
        if (op == DEPOSIT) {
            deposit();
        } else if (op == WITHDRAW) {
            withdraw();
```

```
        } else if (op == INQUIRY) {
            inquiry();
        } else {
            System.out.println("잘못된 입력입니다.");
        }
    }

    private void deposit() {
        // 입금 처리를 수행하는 코드
        System.out.println("입금 처리가 완료되었습니다.");
    }

    private void withdraw() {
        // 출금 처리를 수행하는 코드
        System.out.println("출금 처리가 완료되었습니다.");
    }

    private void inquiry() {
        // 조회 처리를 수행하는 코드
        System.out.println("잔액 조회가 완료되었습니다.");
    }
}

------------ 틀린 코드 ------------
public class BankingSystem {
    public static void main(String[] args) {
        int operationCode = 3; // 초기값으로 3을 대입합니다.

        if (operationCode == 1) {
            depositMoney();
        } else if (operationCode == 2) {
            withdrawMoney();
        } else if (operationCode == 3) {
            checkBalance();
        } else {
            System.out.println("잘못된 입력입니다.");
        }
    }

    public static void depositMoney() {
        System.out.println("예금을 합니다.");
    }
```

```
    public static void withdrawMoney() {
        System.out.println("출금을 합니다.");
    }

    public static void checkBalance() {
        System.out.println("잔액을 조회합니다.");
    }
}
```

리팩토링 - 주사위 게임 2

```
int diceNumber1 = (int)(Math.random() * 6 + 1);
int diceNumber2 = (int)(Math.random() * 6 + 1);
int diceNumber3 = (int)(Math.random() * 6 + 1);
int diceNumber4 = (int)(Math.random() * 6 + 1);
```

이렇게 반복되는 것들이 있으니 하나로 묶고 싶어짐 -> for 문과 배열 이용

```
int[] diceArrayNumber = new int[4];
for(int i = 0; i < diceArrayNumber.length; i++) {
 diceArrayNumber[i] = (int)(Math.random * 6 + 1);
}
```

완성 여기서 숫자들은 따로 선언을 해주어 변수로 만들어주기

도메인 생각의 흐름

먼저 문제를 읽거나 주제에 대해 이거부터 해볼까로 시작

- 주사위 게임에서는 사용자와 주사위 굴리는 것이 중점
- - 사용자 클래스 생성
- - 주사위 클래스 생성
- 그리고 점수에 관한 정보도 필요하겠구나
- - 점수 클래스 생성

시작하면 그 작은 Domain 에만 집중할 것

- 다른 클래스에 대해선 생각하지 말 것

printf()

```
public class Printf {

    public static void main(String[] args) {

        int age = 10;
        String addr = "경기도";

        //'%n'지시자를 사용 안 했을 경우
        System.out.printf("줄바꿈 기능");
        System.out.printf("없음.");

        //'%n'지시자를 사용한 경우
        System.out.printf("줄바꾸기%n");
        System.out.printf("성공!%n");

        System.out.printf("내 나이는 %d 살 입니다.%n",age);
        System.out.printf("내 나이는 %d 살 이고 %s 에 살고
있습니다.%n",age,addr);

    }//main

}//class
```

- %n: 줄바꿈 기능
- %d: 정수형식으로 출력
- %s: 문자열 형식으로 출력
-

실행결과

```
줄바꾸기 기능없음.줄바꾸기

성공!

내 나이는 10 살 입니다.

내 나이는 10 살이고 경기도에 살고 있습니다.
```

toString()
객체가 가지고 있는 정보나 값들을 문자열로 만들어서 리턴하는 메소드

루프에 대한 이해
for 문 -> 참이면 계속 수행문을 수행, 조건식이 거짓이 되면 stop
여기서 출력을 하면 참이 되는 만큼 계속 출력된다는 것을 명심하자!

배열

StackArrayTest
HeapArrayTest

```
public class Summary7 {
    public static void main(String[] args) {

        int NUMBER = 5;
        int START = 0;

        // 배열 생성
        // 배열 만드는 방법
        // 1. 데이터 타입 적고 대괄호[] 적어주기
        // 2. 대괄호 옆에 변수 이름 작성
        //    - 당연히 직관성 있도록
        // 3. 필요하다면 중괄호{} 열고 초기화
        //    또는 new 연산자를 이용
        //    - new 를 통해 할당하는 경우에는 Heap 에 할당

        int[] numberArray1 = {1, 2, 3, 4, 5}; // Stack 은 중괄호{} 내에서
사용 - StackArray
        int[] numberArray2 = new int[NUMBER]; // Heap 은 new 연산자를 이용할
때 사용 - HeapArray

        // 배열의 전체 길이 파악하는 법 - numberArray.length 형태
        for (int i = START; i < numberArray1.length; i++) {
            // 배열은 0 부터 시작
            // numberArray1[0] = 1
            // numberArray1[1] = 2
            // numberArray1[2] = 3
            // numberArray1[3] = 4
            // numberArray1[4] = 5
```

```
            // 위 형태로 동작하여 for 루프에서
            // i 값이 증가함에 따라 모든 배열의 원소들을 출력
            System.out.println("배열 출력: " + numberArray1[i]);
        }

        for (int i = START; i < numberArray2.length; i++) {
            numberArray2[i] = i + 1;
            System.out.println("배열 출력: " + numberArray2[i]);
        }
    }
}
```

For 문

```
public class Summary8 {
    public static void main(String[] args) {
        // for 문 작성하는 법
        // 1. for 옆에 소괄호() 옆에 중괄호{}
        // 2. 소괄호 내부구성
        //    (초기화; 조건 ; 증감)
        // 3. 중괄호 내에 조건이 참일 때 수행할 수행문

        int sum = 0;
        int count = 0;

        for( int i = 0; i < 2; i++) {
            System.out.println("안녕 반가워");

            // sum 을 구하는 경우
            // 먼저 sum 변수 초기화
            sum = sum + i;
            // == sum += i
            // sum 이 i 만큼 누적된다

            System.out.println("count = " + (count++) + ", sum = " + sum);
            // 지금 이상태로 쓰면 번갈아가면서 출력됨
            // - 출력 결과 -
            // 안녕 반가워
            // count = 0, sum = 0
```

```
            // 안녕 반가워
            // count = 1, sum = 1
            // 유의할 것!
        }
        System.out.println("주어진 식의 합 = " + sum);
    }
}
```

IF 문

```
public class summary1 {
    // if 문 작석 방법
    // 1. if 작성 하고 소괄호() 작성하고 중괄호 {}작성
    // 2. 소괄호 안에는 조건문 작성
    // 3. 중괄호 안에는 조건이 일치한다면 수행할 수행문 작성

    public static void main(String[] args) {

        final int PERMIT_AGE = 18;
        int inputAGE = 15;

        if (PERMIT_AGE < inputAGE) {
            System.out.printf("입장 가능합니다.");
        } else {
            System.out.println("입장 불가능합니다.");
        }

        // else if 보다는 if 를 여러번 쓰는 것이 가독성 측면에서 추천

        final int PERMIT_KIDS = 13;

        if (PERMIT_AGE < inputAGE) {
            System.out.println("성인용입니다.");
        }
        if (PERMIT_KIDS < inputAGE) {
            System.out.println("아동용입니다.");
        }
    }
}
```

While 문

```
public class summary3 {
    public static void main(String[] args) {
        // While 문 작성 방법
        // 1. while 적고 소괄호() 적고 중괄호{}
        // 2. 소괄호 내부에 조건식 작성
        // 3. 중괄호에는 조건이 참일 경우 수행할 수행문 작성

        int idx = 0;
        final char ch = 'A'; // 캐릭터형으로 문자 A를 입력

        while (idx < 10) {
            System.out.println("idx = " + (idx++) + ", 안녕" + (char) (ch +
idx));
        } // idx++ 이라고 하지 않으면 무한루프를 돌게 됨
          // idx = 0이기 때문에 계속 10보다 작기 때문에
    }
}
```

Switch Test

```
import java.util.Scanner;

public class summary2 {
    public static void main(String[] args) {
        // Scanner: 키보드 입력 처리를 위해 사용하는 객체
        // System.in: 입력 시스템을 의미
        // 입력 장치에 해당하는 키보드를 의미한다

        // 사람의 키보드 입력을 받고 싶다면 아래 코드 한 줄을 입력하세요
        Scanner scan = new Scanner(System.in);

        // boolean: 참/거짓을 표현하는 자료형
        boolean isloop = true;

        // isloop가 true인 동안 계속 반복
        // 와중에 키보드 입력을 받는다
        while(isloop) {
            System.out.println("숫자를 입력하세요.");
```

```
            int inputNumber = scan.nextInt();
            // 키보드 입력으로 int 타입을 수신하다면 nextInt()를 사용
            // double 타입 - nextDouble(), float 타입 - nextFloat()

            // Switch 문 작성 방법
            // 1. switch 적고 옆에 소괄호() 적고 옆에 중괄호{} 적기
            // 2. 소괄호 안에는 switch case 에서 사용할 조건 작성
            //    -> 현재 케이스에서 inpuNumber 는 숫자
            //       case 0 == 입력한 숫자가 0 이라면
            //       case 1 == 입력한 숫자가 1 이라면
            // 3. 중괄호 내부에는 case 조건들을 적고
            //    각 조건들에 대응하는 수행문 작성
            switch (inputNumber) {
                case 0:
                    System.out.println("종료");
                    isloop = false;
                    break; // 루프 빠져나오기
                    // 숫자 0 을 치는 경우 isloop 를 false 로 바꿈 -> 루프
빠져나오게 됨

                case 1:
                    System.out.println("입금!");
                    break;

                case 2:
                    System.out.println("출금!");
                    break;

                case 3:
                    System.out.println("조회!");
                    break;

                default: // 0~3 사이의 숫자가 아닌 경우
                    System.out.println("그런 명령은 존재하지 않습니다!");
                    break;
            }
        }
    }
}
```