1주차 학습 요약 정리

1. 변수 (Variable)

1. 변수의 타입

- 변수(variable)란 데이터(data)를 저장하기 위해 프로그램에 의해 이름을 할당받은 메모리 공간을 의미 한다.

- 정수형 변수는 char형, int형, long형, long long형 변수로, 실수형 변수는 float형, double형 변수로 구분 된다.

2. 변수를 만드는 방법

2-1. 데이터 타입을 적는다. (int, float, double, long)

int: 정수형 (숫자 - 음수, 양수) 정수값을 저장한다.

long: 비슷함

float: 소수점 실수값을 저장한다.

double: 조금 더 정밀한 소수점 (실제 작업하면서 double 사용할 일은 거의 드믈다)

char: 문자를 저장하고, 변수 당 하나의 문자만을 저장한다.

boolean: 논리 (true, false) 조건식과 논리적 계산에 사용한다.

2-2. 변수 이름을 작성한다 (이름은 우리가 만들고 싶은대로 만들 수 있다)

2-3. 필요하다면 초기화를 진행한다.

2-4. 선언 방법 : 예) int score;, int score = 100;, char ch = 'A';

3. 결론적으로 변수를 만들 때 가장 중요한 것은 변수 이름인데 명시성을 통해 함께 작업하는 팀원들에게

이것이 무엇을 의미하는 것인지 명확하게 전달하기 위한 목적이 가장 중요하다.

```
int appleCount = 3;

int grapeCount = 5;

int totalCount = appleCount + grapeCount;
```

4. 문자열 + 숫자의 경우엔 앞에 문자열이 나왔기 때문에 자동으로 숫자를 문자열 처리하여 화면에 출력한다.

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

5. 변수 앞에 final을 사용하는 이유: 1. 우선 상수로 고정시킬 수 있다는 이점이 있다.

5-1. 상수로 고정시킬 수 있다는 것의 이점이란 무엇인가 ?

아래 있는 [비교 대상] 을 보면 3.3f를 직접 기입하고 있으므로 향후 프로그램이 커지면 직접 3.3f를

작성한 부분을 모두 찾아서 TAX 변경에 따라 모든 코드를 수정해야 하는 불편함이 발생합니다.

반면 TAX 상수에 숫자를 기입하고 이 상수를 사용한다면 변동 상황이 발생할 때 해당하는 TAX 수치값만

한 번 변경하면 모든 작업이 일괄처리 되므로 편합니다.

```
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);
```

[비교 대상]

```
System.out.println("프리랜서 세전 수입: " + income + ", 세후: " + income * (FULL_PERCENT - 3.3f) / FULL_PERCENT);
```

2. 불변 객체 (Immutable Object)

클래스를 인스턴스화 하여 객체를 만들었고 이것이 불변이라면 무엇이 좋을까요 ?

TAX = TAX + 4;

위 코드는 TAX가 final 이기 때문에 새로운 값을 대입하거나 덧셈, 뺄셈 등등이 불가능합니다.

결론적으로 입력되는 값을 변경하지 못하게 막음으로서 원래 동작해야 하는 동작의 무결성을 보장하게 됩니다.

-> 변경하는 행위 자체를 차단한다.

6. 형변환 (Casting) : 값의 타입을 다른 타입으로 변환하는것이다.

int -> char 변환

(char) 65 수식

'A' 로 출력

2. 조건문과 반복문

조건문 : if문과 switch문 2가지이다.

여러개의 상태중 원하는 하나의 상태만을 동작시키고자 할때 사용한다.

if문이 주로 사용되며, 경우의 수가 많을 때 switch문을 사용한다.

1. if 문

- else는 if문처럼 혼자서 독립적인 사용은 불가능하다.

else는 반드시 하나의 if와 pair로 구성되어야 한다.

else는 if문의 아래에 위치해야 한다.

else는 if문 하나에 하나의 else만 사용이 가능하다.

else는 if문의 조건식이 false일 경우 else구문이 동작된다. if가 false일 경우 무시되고 else가 동작

된다.

if 문을 만드는 방법

1. if를 작성하고 소괄호 () 를 작성합니다.

2. 소괄호 내부에 조건식을 작성합니다.

3. 중괄호 내부에는 조건이 만족된 경우 동자할 코드를 작성합니다.

```
if (PERMIT_AGE < inputAge) {
System.out.println("입장 가능하십니다!");
} else {
System.out.println("입장 불가능하십니다!");
}
```

만약 if, else if, else if, else if 형태로 코드가 작성되면 조건식을 첫번째 if가 만족되지 않았을때 else if 를 보게 되므로

기본적으로 해당 else if에서는 if 의 조건 또한 만족하지 않음을 내포하게 됩니다.

그리고 그 다음 else if에서는 맨 처음 if가 만족하지 않고, 그 다음 else if를 만족하지 않고 그리고 현재의 else if 조건을

만족해야 합니다.

그러므로 depth (깊이)가 깊어질수록 코드를 파악하기 위한 혼동이 가중된다는 문제가 있습니다.

이와 같은 이유 때문에 코드를 작성할때 if, else if, else if 보다는 그냥 if, if, if 가 더 좋습니다.

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

[비교 대상]

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

2. Switch 문

- witch문은 if문과 같이 조건 제어문에 속한다. 하지만 if문처럼 조건식이 <,<=,>,>=와 같은 이상, 이하,초과,미만같은

부등식이 사용될 수 없다. if문은 조건식이 true일 경우에 블록이 실행된다고 하면 switch문은 비교할 변수가

어떤값을 가지냐에 따라 실행문을 선택된다. 오직 ==만 비교할 수 있다. 그러므로 모든 switch문은 if문으로 바꿀 수

있지만 if문에서 부등식이 사용된 경우에는 switch문으로 치환할 수 없다.

- if문 조건식과 달리, 조건식의 계산결과가 int타입의 정수와 문자열만 가능하다.

case :

switch( ) 빈칸에 구동 변수가 입력 되어야하고 이 구동 변수를 각 case : 와 비교하여 만약 에 맞다면 아래의 명령식을 실행 시킨다. 그리고 만약 각 case : 에서 같은 변수값이 존재하지 않는다면 default 명령식을 실행시킨다.

break :

switch문에서 break의 역활은 컴파일러에 이 switch문은 실행 되었다고 맞침표를 찍어주는 일을 한다.

→ break가 없는 코드는 switch문이 끝났다는 명령을 컴파일러에게 말해주지 않았기 때문에 위의 case의 명력식들도 같이 출력 된다.

default :

사용자 입력으로 입력되었던 값이 아닌 다른 수를 입력되면 default 값이 출력된다.

→ switch문에서 default의 역활은 switch문이 조건에 맞지 않는 구동 변수을 받을때 default 값을 출력한다.

→ 생략이 가능하다.

switch 문을 작성하는 방법
1. switch를 적고 소괄호() 를 작성하고 중괄호{}를 작성합니다.
2. 소괄호 내부에 switch case 에서 사용할 조건을 적습니다.
   -> 현재 케이스에서 inputNumber 는 숫자이므로
      case 0: 의 의미는 '입력된 숫자가 0이면' 이라는 뜻을 가집니다.
      case 1: 은 '입력된 숫자가 1이면' 이란 뜻입니다.
3. 중괄호 내부에는 case 조건들을 적고
   각 조건에 대응하는 코드를 작성하면 됩니다.

```
switch (inputNumber){
        case 0:
            // 숫자 0이 들어오는 경우 isLoop를 false(거짓)으로 바꿈
            System.out.println("종료");
            isLoop = false;
            break;

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

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

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


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

```
                    break;

                }
```

반복문 : 종류는 for, while, do while이 3가지가 있다.

동일한 코드를 여러번 반복해서 동작시켜줄때 사용한다.

정수 하나를 초기화하기 위해서 초기화문을 사용하고, 검사하고, true라면 내가 할일을 실행한다.

3. For 문

- 단순 반복 while문 / 가독성 좋은건 for문

- 원하는 상황이 되면 break를 이용해서 반복문을 빠져나갈 수 있다. for, while, do while

셋 모두 break를 통해서 빠져나갈 수 있다.

-for문의 초기값에서 변수를 선언하여 사용할 수도 있다. 단, 여기서 선언된 변수는 for문 안에서만 사용이 가능하다.

for 문을 만드는 방법
1. for 를 작성하고 소괄호()를 작성후 중괄호 {} 를 작성한다.
1. 소괄호 내부는 아래와 같이 구성됩니다.
    (초기화; 조건; 증감)
    - 여기서 초기화란 for문을 최초로 만나는 순간에만 동작하게 됩니다.
    그러므로 없어도 됩니다.
    - 조건은 while if switch 등에서 봤던 조건식과 동일합니다.
    조건을 만족하는 동안 for문이 반복됩니다.
    - 증감의 경우에도 없어도 됩니다.
    표현을 조금 더 예쁘게 만들어주기 위해 증감파트가 존재한다 봐도 무방합니다.
3. 중괄호 내에는 for 문을 반복하며 작업할 내용을 적어줍니다.

continue : or문이나 while문의 {}안에서 continue 문장을 만난 순간 continue문 아래에 있는 실행해야 하는 문장들을 건너 뛰고, 다음 반복을 시작한다.

중요한 것은 어찌되었든 for문은 조건 파트가 참인 동안은 언제든지 반복된다는 것입니다.

```
for (; index < END;) {
		System.out.println("index = " + index++);
 }
```

루프를 돌면서 덧셈을 하려면 반드시 아래아 같이 외부에 0으로 초기화된 변수를 가지고 누산해야 합니다.

```
int sum = 0;
int count = 0;
for (int idx = START; idx <= END; idx++) {
		// sum += idx;
		// sum은 sum + idx라는 왜 값이 누산이 되는거야?
		// '=' 연산자는 '~과 같다가 아닙니다.'
		// 오른쪽에 있는 정보를 왼쪽에 대입합니다.
		// '=' 연산자는 대입 연산자
		sum = sum + idx;
		System.out.println("count = " + (++count) + ", sum = " + sum);
}
		System.out.println("3 ~ 10까지의 합: " + sum);
```

조건이 없으므로 무조건입니다.

```
for (;;) {
		System.out.println("무한 반복");
}
```

짝수 출력하기

```
 for (int i = START; i <= END; i++) {
		// '%' 연산자는 나머지 연산자 입니다.
		// (mod 2)와 동의어이며 이건 몰라도 됩니다.

		// 현재 i 값을 2로 나눈 나머지가 0이냐를 묻는것이고
```

```
                // 2로 나눈 나머지가 0이란 것은 짝수임을 의미합니다.
                // 고로 이 로직은 3 ~ 10 사이의 홀수만 출력합니다.
                if (i % 2 == 0) { continue; }   // continue 는 skip과 동일합니다.

                System.out.println("i = " + i);
}
```

4. While 문

- 내부의 조건이 true라면 {} 내부의 코드를 실행한다. 무한 반복 코드가 된다.

While 문 작성 방법

1. 일단 while을 적고 소괄호() 를 작성하고 중괄호{} 를 작성합니다.
2. 소괄호 내부에 조건식을 작성합니다.
   <<<--- idx < 10의 엄밀한 뜻은 아래와 같습니다.
   idx 변수의 값이 숫자 10보다 작은게 맞니? 라고 물어보는것입니다.
   그리고 그 답으로 참(true) 혹은 거짓 (false) 가 튀어나옵니다.
   while(true) 혹은 while(false) 로 치환되므로
   조건이 만족되며 루프를 돌고 만족되지 않으면 루프를 빠져나오게 됩니다.
3. 중괄호 내부에 조건이 만족되는 동안 반복시킬 코드르 작성합니다.

```
 while (idx < 10) {
                System.out.println("idx: " + idx + ", 안녕:" + (char)(ch + idx));
                // char 타입의 변수들은 독특한 특성을 가지고 있습니다.
                // ASCII 코드 특성인데 실제 알파벳 'A' 는 숫자 65에 해당됩니다.
                // 이와 같은 이유로 실질적으로 숫자값들과 덧셈, 뺄셈등의 연산을 수행할 수 있습니다.
                // 위 규칙에 따르면 'B' 는 66이라는 것도 알 수 있습니다.
                idx++;
}
```

3. 배열 (Array)

1. 배열(Array)이란?

- 같은 타입의 여러 변수를 하나의 묶음으로 다룬다.

- 많은 양의 값을 다룰 때 유용하다.

- 배열의 요소는 연속적이다.

2. 배열의 선언 (초기화)

배열의 초기화 final int[] numberArray = { 1, 2, 3, 4, 5 };

- 배열은 0부터 시작한다.

- 10개짜리 배열을 가지고 있다. (stack영역에 int배열 5개 들어있는것. 0~ 4까지)

3. 배열이름.length 는 배열의 길이를 알려준다.

배열의 경우 아래와 같이 numberArray.length 형태로 전체 길이를 파악 할 수 있습니다.

```
for (int i = START; i < numberArray.length; i++) {
		// 배열이라는 녀석은 메로리 공간상에 순차적으로 배치됩니다.
		// 메모리 공간이라는 것이 생소하다면
		// 박스들이 일렬로 나열되어 있는 상황을 생각해도 됩니다.

		//   0     1     2     3     4        <<-- 5개 있지만 0 ~ 4로 표현됨
		// [   ][   ][   ][   ][   ]

		// 주의할 부분이라면 배열의 시작이 0부터라는 것에 주의를 해주세요.
		// 우리는 1부터 100까지라고 말하지만
		// 배열은 0부터 99까지 움직이게 됩니다.

		// final int[] numberArray = { 1, 2, 3, 4, 5 };
		// 위쪽에 이 코드는 아래와 같이 배치되어 있습니다.
```

```
        //    0      1      2      3      4
        // [ 1 ][ 2 ][ 3 ][ 4 ][ 5 ]

        // 그러므로 numberArray[0]은 숫자 1을 표현하고
        // numberArray[1]은 숫자 2를 표현하며
        // numberArray[2]은 숫자 3
        // numberArray[3]은 숫자 4
        // numberArray[4]은 숫자 5

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

배열 만드는 방법

1. 데이터 타입을 적고 대괄호[] 를 적습니다.

2. 변수 선언하듯 변수 이름을 작성합니다.

    당연히 이름이기 때문에 전달력 및 표현력이 중요합니다.

3. 필요하다면 중괄호를 열고 초기화를 하거나 또는 new 데이터타입(1번) [] 형태로 heap에 메모리 할당을 강제할 수 있습니다.

여기서 이야기 한 Heap과 대조되는 것이 Stack 입니다.

현재 위 코드는 Stack이라는 지역 변수에 설정하는 배열이며 new를 통해 할당하는 경우엔 Heap이라는 공간에 할당합니다. (new를 하면 Heap영역에 들어간다.)

Heap에 할당된 메모리 변경 금지를 요청한 것이고 내부에 배치하는 것에는 영향을 받지 않기 때문이다.

```
final int[] numberArray = new int[ALLOC_ARRAY_NUMBER];

for (int i = START; i < ALLOC_ARRAY_NUMBER; i++) {
```

```
numberArray[i] = i + 1;

                // printf의 경우 format을 출력한다는 뜻으로 printf 입니다.
                // format은 %d의 경우 정수형(int)
                // %s의 경우(String)인데 필요 없죠 java는 + 가 되니까요
                // %f의 경우엔 (float, double)등을 처리합니다


                // printf("numberArray[%d] = %d₩n", i, numberArray[i])

                // 위 케이스에서 첫 번째 %d와 두 번째 %d가 보입니다.
                // 첫 번째는 ', i' 가 대응해서 i 값이 %d를 대체하게 됩니다.
                // 두 번째는 ', numberArray[i]' 가 대응해서
                // numberArray[i]에 해당하는 배열값이 %d를 대체합니다.

System.out.printf("numberArray[%d] = %d₩n", i, numberArray[i]);

//System.out.println("numberArray[" + i + "] = " + numberArray[i]);

}
```

아래와 같이 새로운 메모리를 할당해서 전달하는 것을 막고 있습니다.

조금 풀어보자면 객체를 상수화 하느냐 객체 내부의 값을 상수화 하느냐의 관점으로 봐야합니다.

현재 관점은 객체를 상수화하였기 때문에 다른 객체 대입이 막히는 모습입니다.

numberArray = new int[8];

그렇다고 궁극적으로 Stack과 Heap의 차이가 무엇인가?

Stack은 앞서서 주사위 문제에서도 봤듯이 Loop { finak intdata } 형태가 있다면 data 변수가 루프마다 초기화되는 것을 볼 수 있습니다. 이런 지역변수 특성을 가지는 녀석들은 죄다 Stack입니다.

반면 new를 해서 Heap에 설정되는 정보들은 메모리에 상주하게 됩니다.

그러므로 언제 어디서든 데이터에 접근할 수 있게 됩니다.

결론: Stack은 중괄호{} 내에서 사용되고, Heap은 new 하고 이후로 사용된다.

for의 변형 버전 foreach 라고도 한다.

foreach 사용법

1. 배열의 데이터 타입을 작성합니다. (여기선 int)

2. 배열의 원소를 표현할 이름을 적당히 지정합니다. (여기선 num)

3. 클론 하나 찍습니다 (:)

4. 정보를 하나씩 꺼내 올 배열을 적어줍니다. (여기선 numberArray)

```
 for (int num: numberArray) {
                System.out.println("배열 원소 출력: " + num);
}
```

비즈니스가 확장됨에 따라 서비스가 커졌고 그에 따라 데이터의 규모도 커졌다. (여기선 배열에 정보가 많다고 가정한다.)

이런 상황에서 for(초기화; 조건; 증감) 과 foreach가 있다면 누가 더 좋을까 ?

첫 번째 이유 -> 초기화나 조건등이 있어 향후 변경에 유리하다 생각됨

두 번째 이유 -> 전자는 계속 조건식을 검사해야함

이런 이유 때문에 사실 데이터를 요청하면 뭉태기로 한 번에 쏘지 않고 필요한 정보만 선별해서 쏘게 됩니다.

가령 AWS에서 1000만명 분량의 데이터를 검색 한 번에 다 가져오게 되면 요금 폭탄맞고 회사 망합니다.

그래서 서비스 관점에선 이런 조각 내기 개념으로서 페이징(paging) 이라는 것을 합니다.

OS 레벨에서 이야기하는 paging 과는 다른 개념이니 주의합시다!

(여기서 이야기하는 paging은 데이터 조각내기 관점이라 보면 됩니다.)

4. 랜덤 함수 Math.Random()

Math.Random()

- Math 클래스에 정의된 난수
- 0.0과 1.0 사이의 double값을 반환한다.
(0.0 <= Math.Random() < 1.0)

랜덤을 만드는 방법

1. Math.random() 을 작성합니다.

2. 최소값과 최대값을 확인합니다.

3. 최소값은 더하기로 표기해주세요 (아래 코드에선 + 1)

4. 최대값은 곱하기로 표기하고 실제 계산은 곱하기하는 값 + 최소값 - 1 입니다. (아래에선 100 + 1 - 1로 100에 해당함)

그러므로 1 ~ 100까지의 숫자를 만듭니다.

```
final int randomNumber = (int) (Math.random() * 100) + 1;       // 1 ~ 100 사이의 난수값이 나온다.

System.out.println("randomNumber = " + randomNumber);

int randomValue = 0;

        for (int i = START; i < END; i++) {
            randomValue = (int)(Math.random() * MAX) + 1; // 1 ~ 77
            System.out.println("i = " + i + ", randomNumber = " + randomNumber);
        }

        System.out.println();
```

```
        // 5 ~ 15 표현
        for (int i = START; i < END; i++) {
            randomValue = (int)(Math.random() * (MAX - MIN + 1 )) + MIN;
            System.out.println("i = " + i + ", randomNumber = " + randomNumber);
        }
```

Math.Random() 을 이용한 주사위 게임 만들기

```
class Game2 {

    final private int MAX_DICE = 4;
    final private Dice3[] diceArray;
    final private Score2 score;

    public Game2() {
        // Stream.generate(Dice3::new)는
        // Dice3 객체의 생성자를 호출함을 의미합니다.
        // .limit(MAX_DICE)를 통해서 생성하는 개수를 4개로 제한합니다.
        diceArray = Stream.generate(Dice3::new).
                limit(MAX_DICE).toList().
                toArray(new Dice3[0]);

        // 위에서 만든 diceArray를 개별적 요소로 분해합니다.
        // Arrays.stream(diceArray).mapToInt(elem 까지의 내용입니다.
        // 여기서 elem은 분해된 각 diceArray의 요소이고
        // 이 요소는 Dice3 객체에 해당하므로
        // elem.getDiceNumber()를 통해 주사위 값을 확보합니다.
        // 이후 만들어진 배열(4개)에서 모든 주사위의 값을 sum()을 통해 합칩니다.
        score = new Score2(
                Arrays.stream(diceArray).
                        mapToInt(elem -> ((Dice3) elem).getDiceNumber()).sum());
    }

    public Boolean checkWin () {
```

```
            return score.checkWin();
        }

        @Override
        public String toString() {
            return "Game{" +
                    "diceArray=" + Arrays.toString(diceArray) + '₩n' +
                    ", score=" + score +
                    '}';
        }
}

class Dice3 {
        final private int MIN = 1;
        final private int MAX = 6;
        final private int diceNumber;

        public Dice3() {
            this.diceNumber = (int) (Math.random() * (MAX - MIN + 1)) + MIN;
        }

        public int getDiceNumber() {
            return diceNumber;
        }

        @Override
        public String toString() {
            return "Dice{" +
                    "diceNumber=" + diceNumber +
                    '}';
        }
}

class Score2 {
        final private int WIN_DECISION1 = 3;
        final private int WIN_DECISION2 = 4;
        final private int totalScore;
```

```
    public Score2(int totalScore) {
        this.totalScore = totalScore;
    }

    @Override
    public String toString() {
        return "Score{" +
                "totalScore=" + totalScore +
                '}';
    }

    public Boolean checkWin () {
        if ((totalScore % WIN_DECISION1 == 0) ||
                (totalScore % WIN_DECISION2 == 0)) {

            return true;
        }

        return false;
    }
}

public class Dice {
    public static void main(String[] args) {
        Game2 game = new Game2();
        System.out.println(game);
        System.out.println(game.checkWin() ? "승리!" : "패배!");
    }

}
```

* CustomRandom 만들기

```
public class CustomRandom {
```

```
    final private static int MIN = 0;
    // static은 언제나 메모리에 상주함 (Stack도 Heap도 아니다)
    // 그러므로 별도로 new를 할 필요 없이 사용할 수 있다.
    // 대표적으로 main, Math.random 같은것
    public static int generateNumber (int min, int max) {
        return (int) (Math.random() * (max - min + 1)) + min;
    }

    public static int generateNumber (int max) {
        return generateNumber(MIN, max);
    }
}
```

5. 스캐너 Scanner

Scanner 란?

- Scanner는 키보드 입력 처리를 위해 사용하는 객체입니다.

사용자 입력을 받기 위해 아래 코드가 사용된다 보면 되겠습니다.

사용자 입력이란 구체적으로 키보드 입력을 의미합니다.

System.in 이란 입력 시스템을 의미하므로 입력 장치에 해당하는 키보드를 의미한다 봐도 무방하겠습니다.

결론: 사람의 키보드 입력을 받고 싶으면 아래 코드 한 줄을 입력하세요.

```
Scanner scan = new Scanner(System.in);
```

Scanner 를 이용한 회원 정보 입력하기

```
Class Member {

static Scanner scanner = new Scanner(System.in);

static class EmailAddress {

    private String EmailAddress;


    public EmailAddress() {
        this.EmailAddress = EmailAddress;
```

```
            System.out.println("이메일을  입력하세요 : ");
            String EmailAddressInput = scanner.nextLine();

            System.out.println(EmailAddressInput);


        }
}

static class Password {
        private Integer Password;

        public Password() {
            this.Password = Password;

            System.out.println("비밀번호를  입력하세요 : ");
            int PasswordInput = scanner.nextInt();

            System.out.println(PasswordInput);
}


    }
}

public class MemberClassHomework {
        public static void main(String[] args) {
            Member member = new Member();
            new Member.EmailAddress();
            new Member.Password();
        }
}
```

6. class 와 객체

1. class 란?

- 생성자는 class의 이름과 같습니다.

그리고 아래와 같이 리턴 타입이 없습니다.

1-1. class 를 만드는 규칙

1. public을 적고 클래스 이름을 적은 이후 소괄호() 를 작성후 중괄호 {} 를 작성합니다.

2. 만약에 외부에서 값을 입력 받을 것이라면 소괄호에 입력받을 형태를 작성합니다.

3. 실제 클래스가 new를 통해 객체화 될때 구동시키고 싶은 작업을 중괄호 내부에 배치합니다.

즉 현재는 new를 통해 만들어질 때 기본값으로 전구를 꺼놓은 상태로 시작하게 됩니다.

```
public Led(Boolean isTurnOn) {
        // this는 Led 클래스 자기 자신임
        // (좀 더 정확히는 new로 생성된 객체 자신)
       this.isTurnOn = false;
       System.out.println("생성자 호출");
}
```

매서드를 작성하는 방법 -

클래스 내부에 기능을 수행하는 집합들을 매서드라고 부릅니다.

1. public 을 작성하고 소괄호()를 작성하고 중괄호를 {} 작성합니다.

2. 리턴 타입을 public 옆에 작성합니다.

3. 매서드의 이름을 그 옆에 작성해줍니다.

    이름은 역시나 이 작업이 무엇을 하는지 명시적으로 알려줄 수 있는 형태가 좋습니다.

4. 중괄호 내에서는 실제 매서드 이름에 해당하는 작업을 진행하면 됩니다.

    이 때 단순히 작업만 하고 정보 반환이 없다면 리턴 타입은 void 입니다.

    참/거짓을 반환한다면 Boolean(Boolean) 입니다.

    숫자등등이라면 Long(Long) 이나 Integer(int) 입니다.

    무엇을 반환(리턴) 하느냐에 따라 적절한 형태를 적어주면 됩니다.

* 도대체 리턴 된다는 것은 무엇을 의미 하는가 ?

3 -> [ ] -> 9 (리턴 타입 int or Long)

버튼 누름 -> [ ] -> true (리턴 타입 boolean)

1 -> [ ] -> "예금" (리턴 타입 string)

회원정보 -> [ ] (리턴 타입 void --> 회원정보는 들어있는데 아무것도 없는 경우) == 정확히는 리턴하지 않음을 의미함

[ ] -> 20 (리턴 타입 int)

* 입력 관점에서 살펴봅시다!

3 -> [ ] -> 9 (입력 타입 int)

버튼 누름 -> [ ] -> true (입력 타입 Button class)

1 -> [ ] -> "예금" (입력 타입 int)

[ ] -> 20 (입력 타입 void)

참/거짓 -> [ ] (입력 타입 boolean)

2. Getter 란?

- Getter 는 class 내에서 다루는 정보를 얻기 위해 사용합니다.

- 자기 자신을 지킬 수 없기 때문에 다른 객체의 도움을 받아야합니다.

```
public Boolean getTurnOn() {
return isTurnOn;
}
```

3. Domain 란?

- 수십년간의 SW 전문가들이 이런한 개념들을 정리하였고 그 개념의 일환으로 탄생하게 된 개념이

Domain Service 개념입니다.

Domain Service 를 만들어서 얻는 이점은 비즈니스 관점을 좀 더 명확하게 만들어준다는 이점이 있습니다.

실제로 전구를 키고 끄는 작업을 가지고 비즈니스를 할만한 것들이 많지는 않지만

Domain Service 를 나눠 본다면 아래와 같은 것들이 존재할 것입니다.

키기, 끄기, 깜빡이기

-> 조금 쉽게 접근해 보자면 Domain Service는

객체 스스로가 직접 하기에는 표현이 애매해지는 작업들을 모두 Domain Service로 재배치하게 됩니다.

이를 통해 가독성과 유지보수성의 향상을 가져올 수 있습니다.

제어하는 Controller에 RequestForm 객체를 전달

RequestForm 객체는 Domain Service 에서 처리하기 적합한 형태인 Request로 변환

그리고 Request를 보고 적절한 Entity를 추출하게 되는데 이런 관점으로 접근하면 setter를 완벽하게 제거 할 수 있습니다.

즉 필요하면 final 변수들을 설정해서 전달한다는 의미입니다.

4. Setter 란?

- Setter 는 class 내에서 다루는 정보를 직접 설정하는 목적으로 사용합니다.

```
public void setTurnOn(Boolean turnOn) {
isTurnOn = turnOn;
}
```

5. OOP 란?

OOP (Object Oriented Programmimg)

Domain Driven Development (DDD)

5-1. 잘 만든 OOP란 무엇인가 ?

- OOP란 모든 정보를 객체화하여 레고처럼 필요하면 조립하여 관리하자라는 뜻을 가지고 있습니다.

하지만 모든 정보를 하나의 클래스에서 관리하게 되는 경우 객체가 비대해지면서 해당 객체가 어떤 목적을

가지고 있었는지 목적성을 잃게 됩니다.

Domain이라는 관점은 이렇게 클래스가 어떤 주제에 집중을 하고 있는지를 본다고 생각하면 됩니다.

즉 내가 집중하는 주제가 무엇인가를 알 수 있도록 예쁘게 잘 표현해주는 것을 OOP라 봐도 무방합니다.

전구(LED)를 키는 상황을 생각해봅시다.

Q: 왜 데이터 타입을 적는 곳에 class 이름이 오는 것이지?

A: 클래스라는 것 자체가 커스텀 데이터 타입이기 때문입니다.

- 여러분이 직접 커스텀 할 수 있는 데이터 타입이 클래스라 보면 됩니다.

```
final Led led = new Led(true); // <- 초기 생성(꺼짐 - isTureOn: false)


System.out.println("현재 전구 상태: " + (led.getTurnOn() ? "켜짐" : "꺼짐"));
led.setTurnOn(true); // <- 상태 변경(켜짐 - isTureOn: true)
System.out.println("현재 전구 상태: " + (led.getTurnOn() ? "켜짐" : "꺼짐"));
```