이중 for문(nested for문)

for문 안에 for문이 있는 형태

```
for(int i = 0 ; i < 3 ;  i++) {
        // 반복할 문장
        for(int j = 0 ; j < 3 ; j++) {
                // 반복할 문장
        }
}
```

바깥 for문의 0: 안쪽 012 반복

바깥 for문의 1: 안쪽 012 반복

바깥 for문의 2: 안쪽 012 반복

안쪽에 있는 for문은 총 i횟수 * j횟수 = 9 번 실행

바깥쪽에 있는 i for문은 다음 반복으로 넘어가기 위해서 안쪽 for문의 모든 문장들을 마쳐야 한다.

안쪽의 j for문이 모두 완료 되어야, 바깥의 i for문 입장에서는 한회 반복한 것이 된다.

한 회 반복 시 i값이 변하며 다음 반복으로 넘어가게 된다.

객체지향 언어

객체지향 언어 = 프로그래밍 언어 + 객체지향 개념(규칙)

코드의 재사용성이 높고 유지보수 용이

- OOP (Object Oriented Programming)

1. 캡슐화

2. 상속

3. 추상화

4. 다형성

 # 메소드

이름 뒤 소괄호 안에 있는것

단, 키워드 뒤에 소괄호는 메소드가 아니다 (for, while, if 등)

f (x) = 2x + 1

메소드이름 매개변수 리턴값

- 매개변수란?

외부(호출한 곳)에서 전달한 값을 받아서 메소드로 오게 해주는 중간다리 역할

- 메소드의 목적

1. 코드의 재사용(특정성을 부여해서는 안된다.)

2. 소스코드 간결화로 이해하기가 쉬워진다.

3. 모듈화(하나의 코드/로직을 여러개의 메소드로 분리할 경우 유지보수가 용이하다.)

- 메소드 선언

⑤리턴타입 ①메소드 명 (②자료형 매개변수1, 자료형 매개변수2,...) {

③호출 시 실행할 문장

④return 리턴값;

}

메소드는 **선언부**(⑤,①,②)와 **구현부**(③,④)로 이루어져 있다.

① 동사로 작성 eat( ), wear( ), sum( )

② 외부에서 전달받을 값이 있다면 자료형과 함께 변수를 선언하고,

여러개의 값을 전달받을 경우 콤마로 구분한다.

만약 외부에서 전달받을 값이 없다면 생략 가능하다.

③ 생략이 가능하며, 메소드의 기능을 구현하는 로직(알고리즘)을 작성한다

④ 리턴값은 메소드의 연산처리한 결과값으로 최대 1개까지 반환할 수 있다.

return 을 만나면 메소드를 종료하고 다시 호출한 곳으로 되돌아 가기 때문이다.

만약 외부에 반환할 값이 없을 경우 생략 가능

⑤ 리턴값이 있다면 리턴값의 자료형을 선언, 리턴값이 없다면 void를 선언

- 메소드 선언 순서

Q1) 두 정수의 곱셈 메소드 선언

1. multiply ( ) {}

메소드 이름을 생각하고 소괄호와 중괄호를 붙인다.

2. multiply (int num1, int num2) {}

어떤 매개변수를 받아올지 생각한다(두 정수를 받기 위해 매개변수 2개가 필요)

같은 타입의 매개변수를 여러개 받아와도 자료형은 각각 써줘야한다.

3. multiply (int num1, int num2) {

```
    int result = num1 * num2;
```

}

실행할 문장을 작성한다.

4. multiply (int num1, int num2) {

```
    int result = num1 * num2;
    return result;
```

}

외부로 반환할 리턴값을 작성한다.

5. int multiply (int num1, int num2) {

```
    int result = num1 * num2;
    return result;
```

}

리턴값을 통해 선언부의 리턴타입을 결정한다.

※ 재사용성과 유지보수를 위해 하나의 메소드는 한가지 기능만 수행하도록 작성한다.(리팩토링)

- 메소드 사용

메소드는 클래스 내부에 있으므로 해당 메소드가 속해 있는 클래스 타입으로

먼저 참조변수를 만들어 주어야 한다.

클래스명 참조변수명 = new 클래스명( );

참조변수명.메소드명( );　　// 매개변수가 선언되지 않은 경우

참조변수명.메소드명(값1, 값2...);　　// 매개변수가 선언된 경우

1. 메소드에 매개변수가 선언된 경우에는 개수와 타입에 맞게 값을 넘겨주어야 한다.

2. 리턴타입이 void일 경우, 리턴값이 없기 때문에 사용시 그냥 호출 해도 ok!

만약 리턴타입이 있을 경우, 사용 시 메소드 통째로가 "리턴값" 이기 때문에 값을 저장할 변수를 만들어 준다.

3. 메소드가 수행 중 return을 만나면 실행을 종료 하고 자신을 호출한 곳으로 돌아간다.

Class(반)

서로 관련있는 요소들을 추상적이게 묶어놓은 틀

여러 변수들과 메소드들을 한번에 관리 할 수 있다.

- Class 목적

공통요소를 매번 선언하는 작업이 불편하기 때문에 클래스에 선언하여 필드에 공통요소를 한번만 선언하고

Class를 사용할 때 클래스 타입의 객체를 사용한다.

그래서 하나의 클래스 타입으로 여러개의 "객체"를 생성할 수 있다.(제품설계도 - 제품)

1. 객체화

클래스는 추상적인 개념이라서 바로 사용할 수 없기 때문에 필드들을 구체화 시킬 대상이 필요하다.

그래서 객체화를 통해 구체적인 "객체(instance variable)"를 만든다.

클래스 ----------> 객체(인스턴스)

객체화(instance화)

객체 = 속성(변수) + 기능(메소드)

bungeo1 = 속성(타입, 밀가루, 가격) + 기능(굽기, 먹기)

* 객체 생성을 통해 안에 있는 속성(변수)과 기능(메소드)을 사용할 수 있다.

* bungeo1 은 붕어빵틀 클래스의 객체이다.

* 설계도를 통해 제품을 생성하고 제품을 사용한다.

2. 타입이다.

클래스명 객체명;

BungeoPpang bung1;

클래스를 통해 객체가 만들어지기 때문에 객체의 타입은 해당 클래스이다.

3. 주어이다.

첫글자를 항상 **대문자**로 적는다.

- 클래스 선언

다른 클래스 외부에서 선언

class 클래스명 {

필드(변수선언, 메소드선언)

※ 실행문xxxxxxx(출력메소드)

}

- 클래스 사용

객체 생성을 통해 객체의 변수와 메소드를 사용한다.

①클래스명 ②객체명 = ③new ④클래스명( );

⑤객체명.변수

⑥객체명.메소드( );

① 객체를 만들 클래스타입

② 객체를 다룰 참조변수로 주소값을 저장한다.

③ 객체를 생성하며 주소값을 반환한다.

④ 자동으로 생성자 호출

⑤ 만들어진 객체를 통해 안에 있는 변수를 사용한다.

⑥ 만들어진 객체를 통해 안에 있는 메소드를 사용한다.

- new 연산자

new 연산자를 통해 메모리에 올라가면 주소값을 부여받는다.

그때 할당된 주소값을 참조변수에 저장한다. 따라서 참조변수를 출력하면 주소값이 나온다.

클래스 선언(설계도) > 객체 생성(제품 만들고) > 객체 사용(제품사용)

 # 생성자

new 연산자를 통해서 객체를 생성할 때 반드시 호출이 되는 일종의 메소드

객체생성 시 객체(인스턴스 변수)를 초기화하기 위해 사용한다.

- 생성자 선언

```
클래스이름(자료형 매개변수1, ...) {
    // 주로 객체의 변수를 초기화하는 코드
    // 또는 객체 생성 시 수행할 코드
}
```

1. 생성자는 class 안에 선언하며 이름은 클래스 이름과 같아야 한다.

2. 생성자는 리턴타입이 없고, 리턴을 할 수 없기 때문에 메소드 라고 부르지 않는다.(void 안붙임)

3. **class는 무조건 생성자를 1개이상 가지고 있어야 한다.★★★★★★★**

만약 생성자를 만들지 않았을 경우, 컴파일러가 자동으로 "기본생성자" 만들어 준다.(생략 가능)

하지만 생성자를 직접 선언할 경우, 생성자가 하나 이상 있기 때문에 컴파일러는 기본생성자를 만들어주지 않는다.

- 기본생성자

```
클래스이름 ( ) {};
```

1. 매개변수가 없는 생성자

2. 생성자가 하나도 없을 경우 컴파일러가 자동으로 추가.

3. 기본생성자는 반드시 작성하는것 좋다.