Method

Method

- **수학적 관점에서의 함수(Function)**
- 언어에 따라 함수라고 부르기도 하고 메소드라고 부르기도 한다.

```
f(x) = 3x + 2
g(x) = 4x - 1
f(x) + g(x)
e(x) = f(g(x)) + 1
```

- **Method를 정의하면 얼마든지 쓸 수 있고, 다른 함수의 정의에도 사용할 수 있다.**
- **메소드(method)는 특정 기능의 코드 집합**

- 메소드(method)는 코드를 재사용할 수 있게 해준다.

실생활에서 Method 찾아보기

메소드가 수학적 관점에서 함수라고 한다. 초심자 입장에서 조금 와닿지 않는다. 실생활의 예를 들어보자!

제빵사의 빵만들기

↓ **밀가루**

```
// 반죽기계
void 반죽기(밀가루){
    밀가루를 반죽한다.
}
```

↓ **반죽**

```
// 삽입기계
void 삽입기(반죽){
    반죽에 재료를 삽입한다.
}
```

↓ **재료가 들어간 반죽**

```
// 오븐
void 오븐(재료가 들어간 반죽){
    반죽을 굽는다.
}
```

↓ **빵**

- **반죽기, 삽입기, 오븐** : 특정 기능을 수행하는 장치 → **메소드**
- **(밀가루), (반죽), (재료가 들어간 반죽)** : 각각의 기계에 들어가는 재료 → **입력값 (인자)**
- **반죽, 재료가 들어간 반죽, 빵** : 기능을 수행한 결과물 → **출력값 (반환값)**

Main Method

```
public static void main(String[] args) {
     int i=0;

     While(i<10) {
     System.out.println(i);
          i++
     }
}
```

- 메소드 중괄호 내에 존재하는 문장들이 위에서 아래로 순차적 실행
- **Java Application의 시작은 main Method를 실행하는 것부터 시작된다.**

Define Method

i. **메소드 안에서 메소드를 정의할 수는 없다.**
ii. **클래스 안에서 메소드끼리 서로 내부에서 호출이 가능하다!**

```
class MethodDefAdd {
     public static void main(String[] args) { // 자동실행
          System.out.println("프로그램 시작");
          hiEveryone(12); // hiEveryone Method에 12를 인자로 전달하면서 호출
          hiEveryone(13);
          System.out.println("프로그램 끝");
     }

     public static void hiEveryone( int age ) {
                                   Parameter(매개변수)
                                   호출시 전달받은 인자(Argument)를 메모리 공간에 저장할 변수

          System.out.println("좋은 아침입니다.");
          System.out.println("제 나이는 "+ age +"입니다");
     }
}
```

! **인자와 매개변수의 자료형은 일치해야 한다.**

Define Method - Many Parameter, Non Parameter

```
class MethodDefAdd {
     public static void main(String[] args) {
          double myHeight = 178.0;
          hiEveryone(12, 12.5);
          hiEveryone(13, myHeight);
          byEveryone();
     }

     // 2 Parameter, 전달 순서대로 저장
     public static void hiEveryone(int age, int height) {
          System.out.println("제 나이는 "+age+"입니다");
          System.out.println("제 키는 "+height+"cm 입니다");
```

```
    }

    // Non Parameter
    public static void byEveryone() {
        System.out.println("다음에 뵙겠습니다.");
    }
}
```

---

return(반환)

- 연산문, 메소드가 있던 자리를 연산의 결과값, 메소드가 return하는 값 으로 대체하는것
- 메소드의 종료 (return문을 실행하게 되면 뒤의 나머지 문장 실행하지 않고 종료)

```
class MethodReturns {
    // void : return하지 않겠다.
    public static void main(String[] args) {
        int result = adder(4, 5);
        System.out.println("4와 5의 합 : "+result);
        System.out.println("3.5의 제곱 : "+square(3.5));
    }

    // int : int형 데이터로 return하겠다.
    public static int adder(int num1, int num2)
        int result = num1 + num2;
        return result;
    }

    // double : double형 데이터로 return하겠다.
    public static double square(double num) {
        return num * num;
    }
}
```

메소드의 종료를 위한 return의 활용

```
class OnlyExitReturn {

    public static void main(String[] args) {
        devide(4, 2);
        devide(9, 0);
    }

    public static void divide(int num1, int num2) {
        if(num2==0) {
            System.out.println("0으로는 값을 나눌 수 없습니다.");
            return; // 메소드의 종료만을 의미함
        }
        System.out.println("나눗셈 결과 : " + (num1/num2));
    }
}
```
