

# 탈잉 개발자와 프로그래밍 무작정 배워보기

---

## 3. 메소드와 클래스

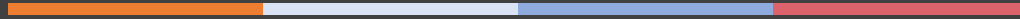
강대규

# 목차



1. 메소드
2. 구구단을 메소드로 만들어보자
3. 클래스, 인스턴스, 생성자
4. 학점계산기 클래스를 만들어보자

# 1. 메소드



# 메소드

---

```
System.out.println("블라블라");  
System.out.println("블라블라");  
System.out.println("블라블라");  
System.out.println("블라블라");
```

```
System.out.println("라블라블");  
System.out.println("라블라블");  
System.out.println("라블라블");  
System.out.println("라블라블");
```

```
System.out.println("블라라블");  
System.out.println("블라라블");  
System.out.println("블라라블");  
System.out.println("블라라블");
```

# 메소드

---

```
System.out.println("블라블라");  
System.out.println("블라블라");  
System.out.println("블라블라");  
System.out.println("블라블라");
```

} 블라블라 4번 출력

```
System.out.println("라블라블");  
System.out.println("라블라블");  
System.out.println("라블라블");  
System.out.println("라블라블");
```

} 라블라블 4번 출력

```
System.out.println("블라라블");  
System.out.println("블라라블");  
System.out.println("블라라블");  
System.out.println("블라라블");
```

} 블라라블 4번 출력

# 메소드

---

```
for (int i = 0; i < 4; i++) {  
    System.out.println("블라블라");  
}
```

```
for (int i = 0; i < 4; i++) {  
    System.out.println("라블라블");  
}
```

```
for (int i = 0; i < 4; i++) {  
    System.out.println("블라라블");  
}
```

# 메소드

---

```
for (int i = 0; i < 4; i++) {  
    System.out.println("블라블라");  
}
```

```
for (int i = 0; i < 4; i++) {  
    System.out.println("라블라블");  
}
```

```
for (int i = 0; i < 4; i++) {  
    System.out.println("블라라블");  
}
```

} 같은 형식의 for 문 반복

# 메소드



뭔가 몇 번이든 사용할 수 있게 하는 건 없을까?



# 메소드



코드를 재사용할 수 있도록 해주는 것

# 메소드 정의와 호출

---

```
void print4times(String text) {  
    for (int i = 0; i < 4; i++) {  
        System.out.println(text);  
    }  
}
```

```
print4times("블라블라");  
print4times("라블라블");  
print4times("블라라블");
```

# 메소드 정의와 호출

---

```
void print4times(String text) {  
    for (int i = 0; i < 4; i++) {  
        System.out.println(text);  
    }  
}
```

메소드 정의

```
print4times("블라블라");  
print4times("라블라블");  
print4times("블라라블");
```

# 메소드 정의와 호출

---

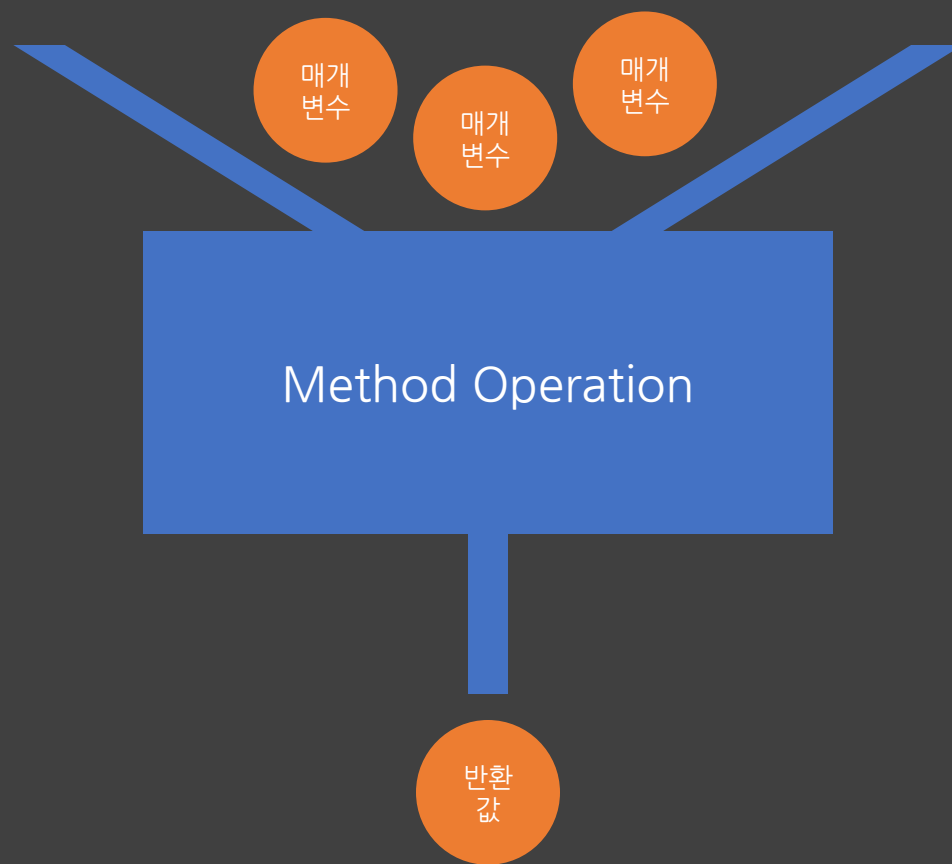
```
void print4times(String text) {  
    for (int i = 0; i < 4; i++) {  
        System.out.println(text);  
    }  
}
```

```
print4times("블라블라");  
print4times("라블라블");  
print4times("블라라블");
```

메소드 호출

# 메소드의 구조

---



# 메소드의 구조

---

메소드 이름

```
void print4times(String text) {  
    for (int i = 0; i < 4; i++) {  
        System.out.println(text);  
    }  
}
```

## 메소드의 구조

---

매개변수, 여러 매개변수가 올 수 있음

```
void print4times(String text) {  
    for (int i = 0; i < 4; i++) {  
        System.out.println(text);  
    }  
}
```

# 메소드의 구조

---

반환 타입, void 는 반환할 값이 없음을 의미

```
void print4times(String text) {  
    for (int i = 0; i < 4; i++) {  
        System.out.println(text);  
    }  
}
```



## 메소드의 구조

---

반환 타입, int 값이 반환된다.

```
int add(int x, int y) {
```

```
    return x + y;
```

```
}
```

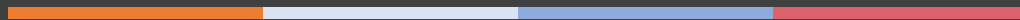
반환할 값.  $x + y$  가 메소드의 결과값으로 반환된다.

# 메인 메소드



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

## 2. 구구단을 메소드로 만들어보자



# 구구단 메소드

---

1. 구구단을 출력한다.
2. 출력할 단 수는 매개변수로 받는다.
3. 반환값은 존재하지 않는다.

## 구구단 메소드

---

```
void printGugudan(int stage) {  
    for (int i = 1; i < 10; i++) {  
        System.out.println(stage + "x" + i + "=" + stage * i);  
    }  
}  
  
printGugudan(3);
```

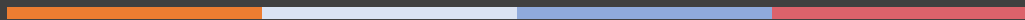
# 학점계산기 메소드

---

학점계산기를 메소드로 만들어보자.

1. 점수를 입력받아 학점을 출력하는 메소드
2. 점수를 입력받아 학점을 return 해주는 메소드

# 전역변수



입력받은 점수를 조작해보자

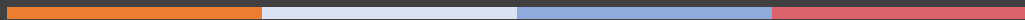
# 전역변수



```
public static void main(String[] args) {  
    int score = 10;  
    System.out.println(score);  
  
    changeScore(score);  
    System.out.println(score);  
}  
  
static void changeScore(int score) {  
    score = 100;  
}
```



# 전역변수



점수가 조작되지 않는다!

# 전역변수

---

`static int score;` 전역변수 (Global Variable)

```
public static void main(String[] args) {  
    score = 10;  
    System.out.println(score);  
  
    changeScore();  
    System.out.println(score);  
}  
  
static void changeScore(int score) {  
    score = 100;  
}
```

### 3. 클래스, 인스턴스, 생성자

---

# 객체지향 프로그래밍



OOP (Object Oriented Programming)

# 객체지향 프로그래밍



OOP (Object Oriented Programming)

→ 프로그래밍을 객체 (“Object”)들의 집합의 관점으로 봄

# 객체지향 프로그래밍



OOP (Object Oriented Programming)

- 프로그래밍을 객체 (“Object”)들의 집합의 관점으로 봄
- 프로그램을 **하나의 완성품**이라고 봤을 때, 객체는 **부품**

# 클래스



변수와 메소드의 집합으로 이루어진 구성체

# 클래스



변수와 메소드의 집합으로 이루어진 구성체

→ 부품의 종류 (설계도)



# 클래스



```
class Calculator {  
  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
  
}
```

# 클래스

---

클래스 이름

```
class Calculator{  
    public int add(int a, int b) {  
        return a + b;  
    }  
  
    public int multiply(int a, int b) {  
        return a * b;  
    }  
}
```

# 인스턴스



클래스를 실체화한 것

# 인스턴스



클래스를 실체화한 것

→ 부품 (실제로 사용이 되는)

## 인스턴스 생성



```
Calculator calculator = new Calculator();
```

# 인스턴스 생성

---

```
Calculator calculator = new Calculator();
```

클래스 이름

객체 이름

## 인스턴스에서 메소드 호출

---

```
Calculator calculator = new Calculator();
```

```
int a = 10;
```

```
int b = 20;
```

```
int c = calculator.add(a, b);
```

# 생성자



객체를 생성할 때 객체를 초기화 해두는 것



# 생성자



아까 만들었던 계산기 클래스에서

객체를 생성할 때 숫자 두 개를 미리 받아보자!

# 생성자

---

```
class Calculator {  
  
    public int x;  
    public int y;  
  
    public Calculator(int a, int b) {  
        x = a;  
        y = b;  
    }  
  
    public int add() {  
        return x + y;  
    }  
  
    public int multiply() {  
        return x * y;  
    }  
}
```

# 생성자

---

```
class Calculator {
```

```
    public int x;  
    public int y;
```

```
    public Calculator(int a, int b) {  
        x = a;  
        y = b;  
    }
```

생성자

```
    public int add() {  
        return x + y;  
    }
```

```
    public int multiply() {  
        return x * y;  
    }
```

```
}
```

## 생성자를 이용한 초기화

---

```
Calculator calculator = new Calculator(10, 20);
```

```
int addResult = calculator.add();  
int multiplyResult = calculator.multiply();
```

## 인스턴스에서 변수 호출

---

```
Calculator calculator = new Calculator(10, 20);
```

```
int x = calculator.x;  
int y = calculator.y;
```

# 생성자와 this

---

```
class Calculator {  
  
    public int x;  
    public int y;  
  
    public Calculator(int x, int y) {  
        x = x;  
        y = y;  
    }  
  
    public int add() {  
        return x + y;  
    }  
  
    public int multiply() {  
        return x * y;  
    }  
}
```

# 생성자와 this

```
class Calculator {
```

```
    public int x;  
    public int y;
```

```
    public Calculator(int x, int y) {
```

```
        x = x;  
        y = y;
```

```
    }
```

????전역변수와 매개변수 이름이 같다?!!!

```
    public int add() {  
        return x + y;  
    }
```

```
    public int multiply() {  
        return x * y;  
    }
```

```
}
```

# 생성자와 this

---

```
class Calculator {  
  
    public int x;  
    public int y;  
  
    public Calculator(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int add() {  
        return x + y;  
    }  
  
    public int multiply() {  
        return x * y;  
    }  
}
```



# 생성자와 this

```
class Calculator {
```

```
    public int x;  
    public int y;
```

```
    public Calculator(int x, int y) {
```

```
        this.x = x;
```

```
        this.y = y;
```

```
    }
```

```
    public int add() {  
        return x + y;  
    }
```

```
    public int multiply() {  
        return x * y;  
    }
```

```
}
```

**Calculator 객체 자신을 의미. 즉 전역변수를 호출**

# 생성자를 여러개 생성

---

```
class Calculator {  
  
    public int x;  
    public int y;  
  
    public Calculator() {  
        x = 10;  
        y = 20;  
    }  
  
    public Calculator(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int add() {  
        return x + y;  
    }  
  
    public int multiply() {  
        return x * y;  
    }  
}
```

# 생성자를 여러개 생성

```
class Calculator {
```

```
    public int x;  
    public int y;
```

```
    public Calculator() {  
        x = 10;  
        y = 20;  
    }
```

생성자1

```
    public Calculator(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }
```

생성자2

```
    public int add() {  
        return x + y;  
    }
```

```
    public int multiply() {  
        return x * y;  
    }  
}
```

## 생성자를 이용한 초기화

---

```
Calculator calculator = new Calculator();
```

```
int addResult = calculator.add();  
int multiplyResult = calculator.multiply();
```

```
Calculator calculator2 = new Calculator(1, 2);
```

```
int addResult2 = calculator2.add();  
int multiplyResult2 = calculator2.multiply();
```

# 생성자를 여러개와 this

```
class Calculator {  
  
    public int x;  
    public int y;  
  
    public Calculator() {  
        x = 10;  
        y = 20;  
    }  
  
    public Calculator(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int add() {  
        return x + y;  
    }  
  
    public int multiply() {  
        return x * y;  
    }  
}
```

중복

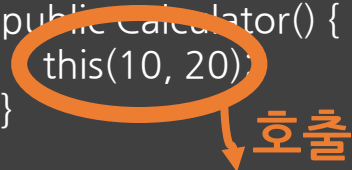
# 생성자를 여러개와 this

---

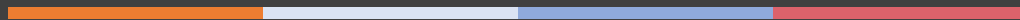
```
class Calculator {  
  
    public int x;  
    public int y;  
  
    public Calculator() {  
        this(10, 20);  
    }  
  
    public Calculator(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int add() {  
        return x + y;  
    }  
  
    public int multiply() {  
        return x * y;  
    }  
}
```

# 생성자를 여러개와 this

```
class Calculator {  
  
    public int x;  
    public int y;  
  
    public Calculator() {  
        this(10, 20);  
    }  
  
    public Calculator(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
  
    public int add() {  
        return x + y;  
    }  
  
    public int multiply() {  
        return x * y;  
    }  
}
```



## 4. 학점계산기 클래스를 만들어보자





# 학점계산기 클래스

---

1. 생성자에서 점수를 받는다.
  - 만약 디폴트 생성자라면 0점으로 초기화해준다.
2. 점수에 따른 학점등급을 반환해주는 메소드가 존재한다.
3. 점수 다시 세팅할 수 있다. (외부에서 점수를 세팅함)