메소드

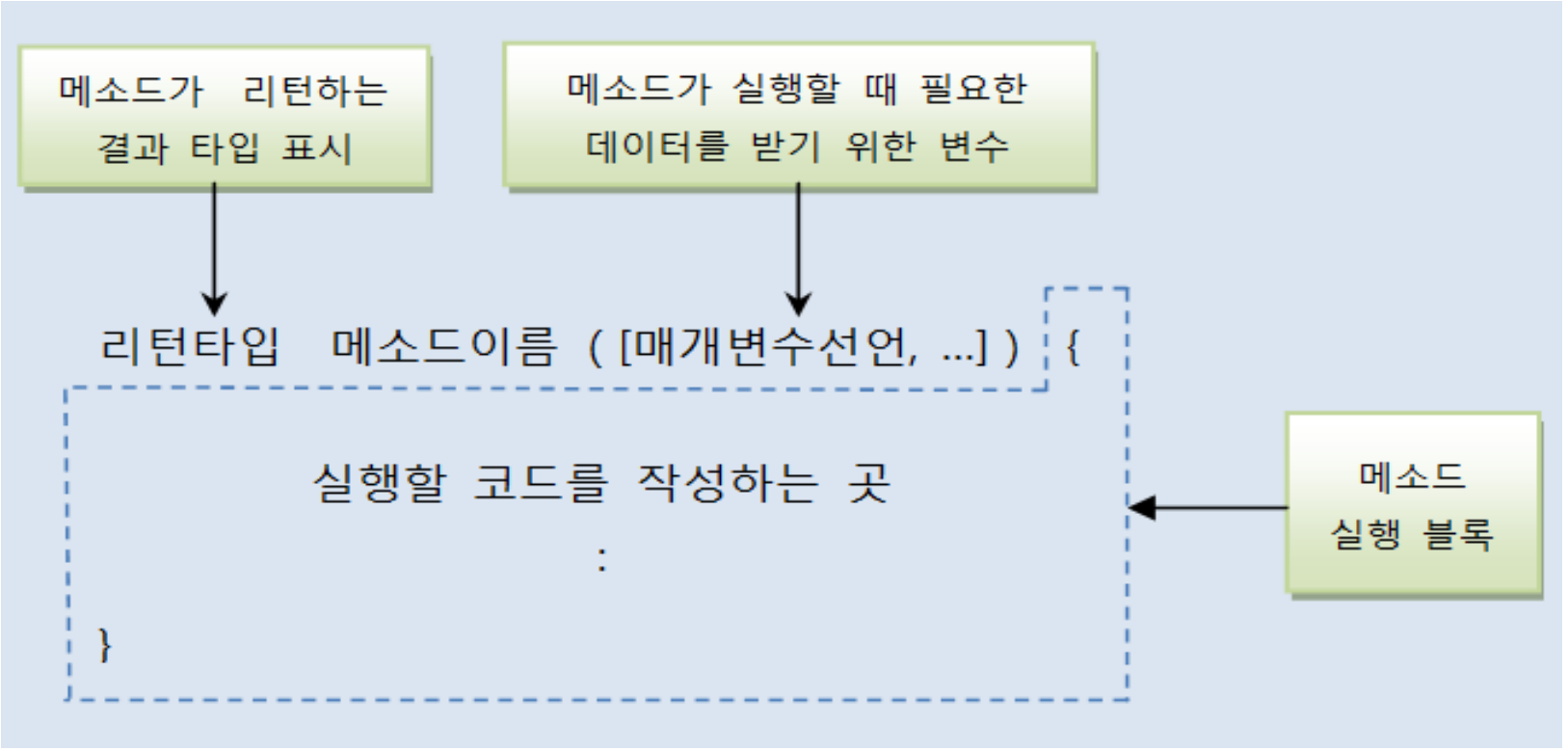
메소드

❖ 메소드란?

- 객체의 동작(기능) 표현
- 호출해서 실행할 수 있는 중괄호 { } 블록
- 메소드를 호출하면 중괄호 { } 블록에 있는 모든 코드들이 일괄 실행

메소드

❖ 메소드 선언


메소드가 리턴하는
결과 타입 표시
메소드가 실행할 때 필요한
데이터를 받기 위한 변수
리턴타입 메소드이름 ([매개변수선언, ...]) {
실행할 코드를 작성하는 곳
:
}
메소드
실행 블록

메소드

❖ 메소드 리턴 타입

- 메소드 실행된 후 리턴하는 값의 타입
- 메소드는 리턴값이 있을 수도 있고 없을 수도 있음
  - `void` 타입

[메소드 선언]

```
void powerOn() { ... }
double divide(int x, int y) { ... }
```

[메소드 호출]

```
powerOn();
double result = divide( 10, 20 );
```

메소드

❖ 메소드 이름

- 자바 식별자 규칙에 맞게 작성
  - 숫자로 시작하면 안 되고, $와 _를 제외한 특수 문자를 사용하지 말아야 한다.
  - 관례적으로 메소드명은 소문자로 작성한다.
  - 서로 다른 단어가 혼합된 이름이라면 뒤이어 오는 단어의 첫머리 글자는 대문자로 작성한다.

  → CamelCase(낙타표기법)

❖ 메소드 매개변수 선언

- 매개변수는 메소드를 실행할 때 필요한 데이터를 외부에서 받기 위해 사용
- 매개변수도 필요 없을 수 있음

[메소드 선언]

```
void powerOn() { ... }
double divide(int x, int y) { ... }
```

[메소드 호출]

```
powerOn();
double result = divide( 10, 20 );
```

```
byte b1 = 10;
byte b2 = 20;
double result = divide(b1, b2);
```

❖ 메소드 선언: Calculator.java

```
public class Calculator {
    // 메소드
    void powerOn() {
        System.out.println("전원을 켭니다.");
    }

    int plus(int x, int y) {
        int result = x + y;
        return result;
    }

    double divide(int x, int y) {
        double result = (double) x / (double) y;
        return result;
    }

    void powerOff() {
        System.out.println("전원을 끕니다");
    }
}
```

❖ 메소드 호출: CalculatorExample.java

```
public class CalculatorExample {
    public static void main(String[] args) {
        Calculator myCalc = new Calculator();
        myCalc.powerOn();

        int result1 = myCalc.plus(5, 6);
        System.out.println("result1: " + result1);

        byte x = 10;
        byte y = 4;
        double result2 = myCalc.divide(x, y);
        System.out.println("result2: " + result2);

        myCalc.powerOff();
    }
}
```

❖ 매개 변수의 수를 모를 경우(다양한 경우)

- 배열을 이용하는 방법

```
int[] values = { 1, 2, 3 };
int result = sum1(values);
int result = sum1(new int[] { 1, 2, 3, 4, 5 });
```

- 가변 매개 변수를 이용하는 방법

```
int sum2(int … values) {  }
int result = sum2(1, 2, 3);
int result = sum2(1, 2, 3, 4, 5);
```

  - 메소드 내에서는 배열로 인식

❖ 매개 변수의 수를 모를 경우: Computer.java

```
public class Computer {
    int sum1(int[] values) {
        int sum = 0;
        for (int i = 0; i < values.length; i++) {
            sum += values[i];
        }
        return sum;
    }

     int sum2(int... values) {

        int sum = 0;
        for (int i = 0; i < values.length; i++) {
            sum += values[i];
        }
        return sum;
    }
}
```

❖ 매개 변수의 수를 모를 경우: ComputerExample.java

```
public class ComputerExample {
    public static void main(String[] args) {
        Computer myCom = new Computer();

        int[] values1 = { 1, 2, 3 };
        int result1 = myCom.sum1(values1);
        System.out.println("result1: " + result1);

        int result2 = myCom.sum1(new int[] { 1, 2, 3, 4, 5 });
        System.out.println("result2: " + result2);

        int result3 = myCom.sum2(1, 2, 3);
        System.out.println("result3: " + result3);

        int result4 = myCom.sum2(1, 2, 3, 4, 5);
        System.out.println("result4: " + result4);
    }
}
```

❖ 리턴(return) 문

- 메소드 실행을 중지하고 리턴값 지정하는 역할
  - return 리턴값;

- 리턴값이 있는 메소드
  - 반드시 리턴(return)문 사용해 리턴값 지정

```
int plus(int x, int y) {
    int result = x + y;
    return result;
}
```

메소드

❖ 리턴(return) 문

- ○ 리턴값이 있는 메소드
  - return 문 뒤에 실행문 올 수 없음

```
int plus(int x, int y) {
    int result = x + y;
    return result;
    System.out.println(result);  // Unreachable code
}

boolean isLeftGas() {
    if(gas ==0) {
        System.out.println("gas가 없습니다.");
        return false;
    }
    System.out.println("gas가 있습니다.");
    return true;
}
```

메소드

❖ 리턴(return) 문

- ○ 리턴값이 없는 메소드
  - return ;
  - 리턴타입은 void

```
void run() {
   while(true) {   // 무한 루프
      if(gas > 0) {
         System.out.println("달립니다.(gas 잔량:" + gas + ")");
         gas -= 1;
      } else {
         System.out.println("멈춥니다.(gas 잔량:" + gas + ")");
         return;   // run() 메소드 실행 종료
      }
   }
}
```

❖ return 문: Car.java

```
public class Car {
    // 필드
    int gas;

    // 생성자

    // 메소드
    void setGas(int gas) {
        this.gas = gas;
    }

    boolean isLeftGas() {
        if (gas == 0) {
            System.out.println("gas가 없습니다.");
            return false;
        }
        System.out.println("gas가 있습니다.");
        return true;
    }
```

❖ return 문: Car.java

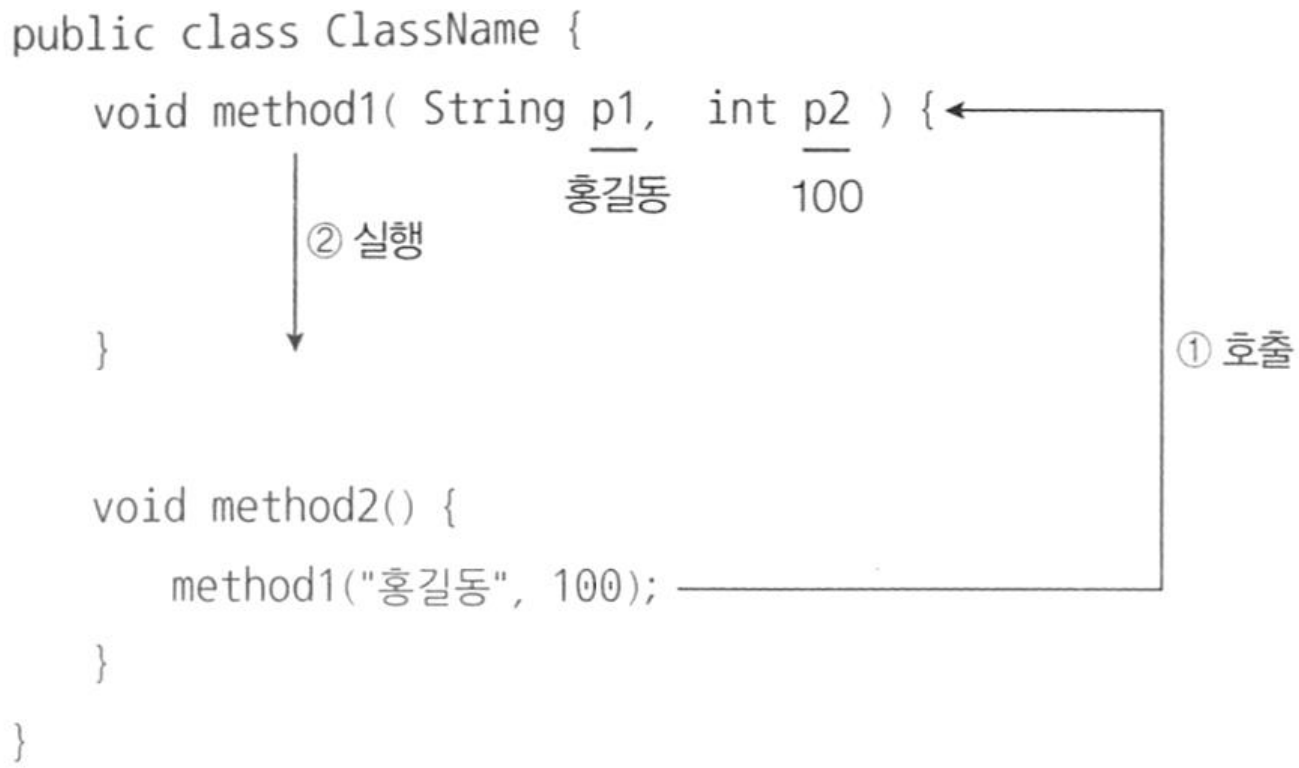
```
    void run() {
        while (true) {
            if (gas > 0) {
                System.out.println("달립니다.(gas잔량:" + gas + ")");
                gas -= 1;
            } else {
                System.out.println("멈춥니다.(gas잔량:" + gas + ")");
                return;
            }
        }
    }
}
```

❖ return 문: CarExample.java

```
public class CarExample {
    public static void main(String[] args) {
        Car myCar = new Car();

        myCar.setGas(5); // Car의 setGas() 메소드 호출

        boolean gasState = myCar.isLeftGas(); // Car의 isLeftGas() 메소드 호출
        if (gasState) {
            System.out.println("출발합니다.");
            myCar.run(); // Car의 run() 메소드 호출
        }

        if (myCar.isLeftGas()) { // Car의 isLeftGas() 메소드 호출
            System.out.println("gas를 주입할 필요가 없습니다.");
        } else {
            System.out.println("gas를 주입하세요.");
        }
    }
}
```

❖ 메소드 호출

- 메소드는 클래스 내·외부의 호출에 의해 실행
- 클래스 내부: 메소드 이름으로 호출



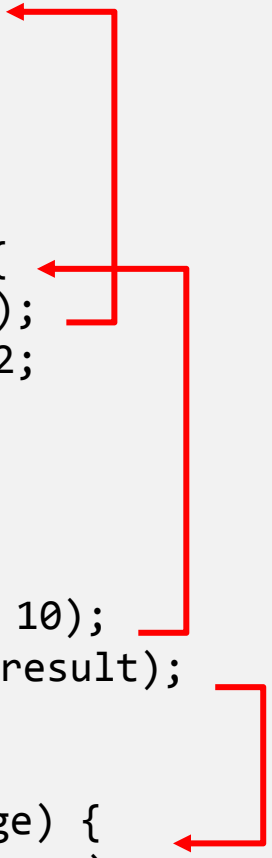
메소드

❖ 메소드 호출

```
public class ClassName {

    int method1(int x, int y) {
        int result = x + y;
        return result;
    }

    void method2() {
        int result1 = method1(10, 20);      //result1에는 30이 저장
        double result2 = method1(10, 20);   //result2에는 30.0이 저장
    }
}
```

❖ 클래스 내부에서 메서드 호출: Calculator.java

```
public class Calculator {
    int plus(int x, int y) {
        int result = x + y;
        return result;
    }

    double avg(int x, int y) {
        double sum = plus(x, y);
        double result = sum / 2;
        return result;
    }

    void execute() {
        double result = avg(7, 10);
        println("실행결과: " + result);
    }

    void println(String message) {
        System.out.println(message);
    }
}
```



메소드

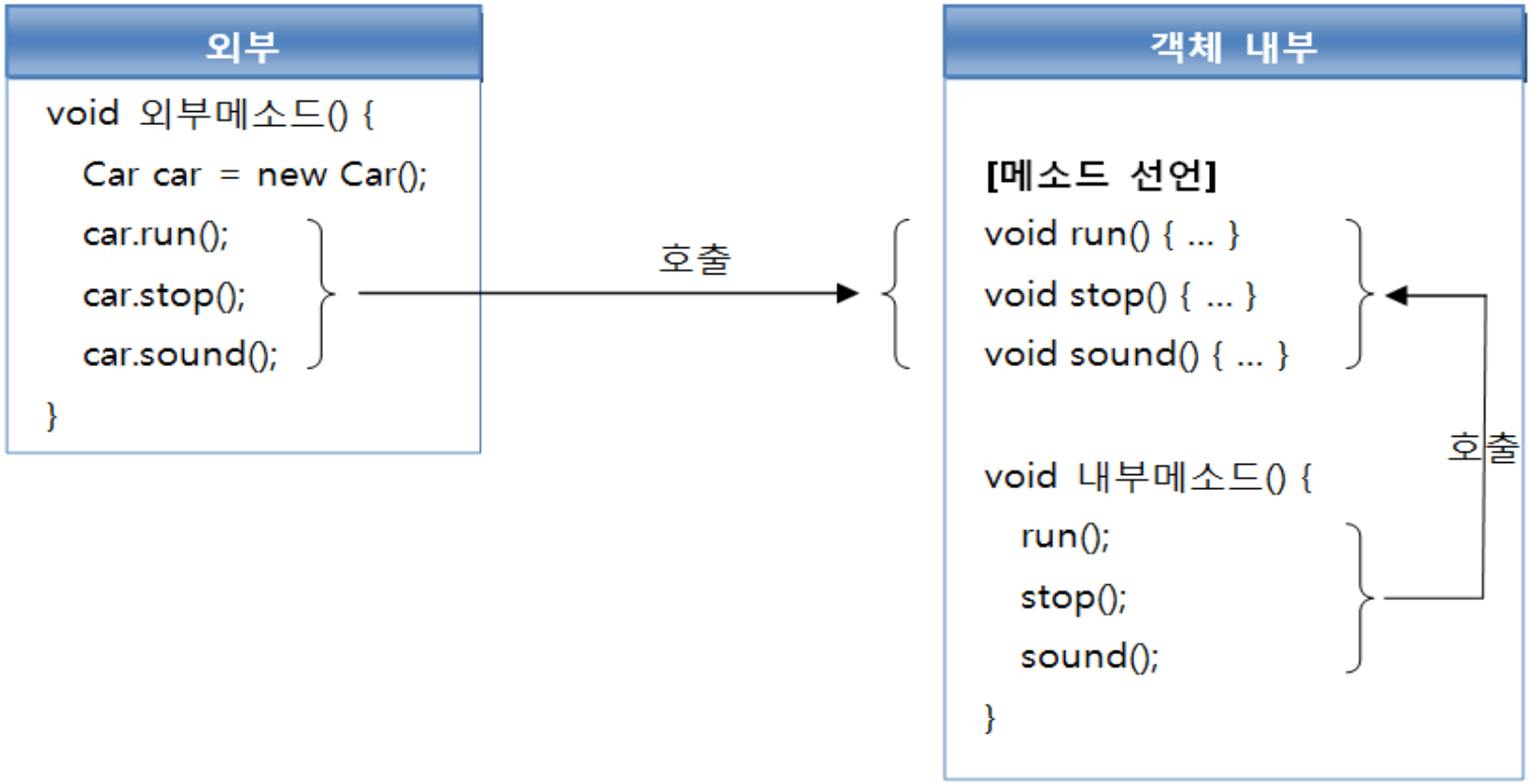
❖ Calculator의 execute() 호출: CalculatorExample.java

```
public class CalculatorExample {
    public static void main(String[] args) {
        Calculator myCalc = new Calculator();
        myCalc.execute();
    }
}
```

메소드

❖ 메소드 호출

- 메소드는 클래스 내·외부의 호출에 의해 실행
- 클래스 외부: 객체 생성 후, 참조 변수를 이용해 호출



메소드

❖ 클래스 외부에서 메서드 호출:Car.java

```
public class Car {
    // 필드
    int speed;

    // 메소드
    int getSpeed() {
        return speed;
    }

    void keyTurnOn() {
        System.out.println("키를 돌립니다.");
    }

    void run() {
        for (int i = 10; i <= 50; i += 10) {
            speed = i;
            System.out.println("달립니다.(시속:" + speed + "km/h)");
        }
    }
}
```

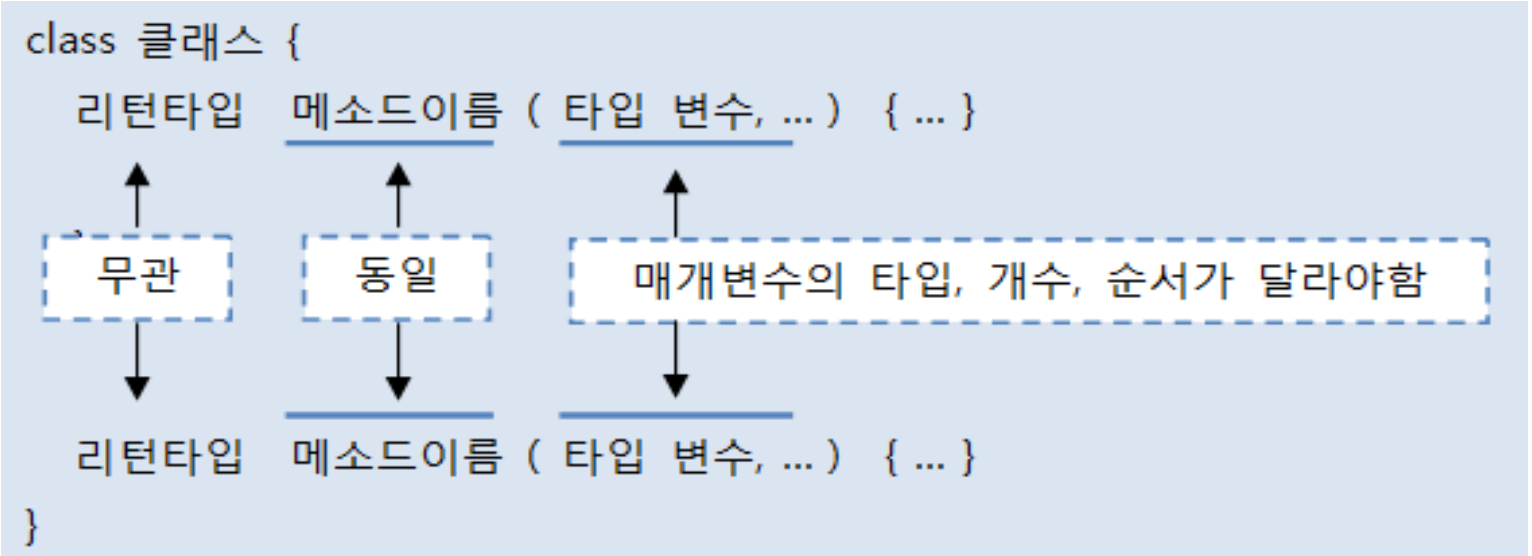
메소드

❖ 클래스 외부에서 메서드 호출: CarExample.java

```
public class CarExample {
    public static void main(String[] args) {
        Car myCar = new Car();

        myCar.keyTurnOn();
        myCar.run();
        int speed = myCar.getSpeed();

        System.out.println("현재 속도: " + speed + "km/h");
    }
}
```

❖ 메소드 오버로딩(Overloading)

- 클래스 내에 같은 이름의 메소드를 여러 개 선언하는 것
- 하나의 메소드 이름으로 다양한 매개값 받기 위해 메소드 오버로딩
- 오버로딩의 조건
  - 매개변수의 타입, 개수, 순서가 달라야 함



❖ 메소드 오버로딩(Overloading)

```
int plus(int x, int y) {
    int result = x + y;
    return result;
}
```

```
plus(10, 20);
```

```
double plus(double x, double y) {
    double result = x + y;
    return result;
}
```

```
plus(10.5, 20.3);
```

- 잘못된 오버로딩

```
int divide(int x, int y) { ... }
double divide(int boonja, int boonmo) { ... }
```

```
int x = 10;
double y = 20.3;
plus(x, y);
```

❖ 메소드 오버로딩(Overloading)

- System.out.println()
  - 대표적인 오버로딩

```
void println() { .. }
void println(boolean x) { .. }
void println(char x) { .. }
void println(char[] x) { .. }
void println(double x) { .. }
void println(float x) { .. }
void println(int x) { .. }
void println(long x) { .. }
void println(Object x) { .. }
void println(String x) { .. }
```

❖ 메소드 오버로딩: Calculator.java

```
public class Calculator {
    // 정사각형의 넓이
    double areaRectangle(double width) {
        return width * width;
    }

    // 직사각형의 넓이
    double areaRectangle(double width, double height) {
        return width * height;
    }
}
```

❖ 메소드 오버로딩: CalculatorExample.java

```
public class CalculatorExample {
    public static void main(String[] args) {
        Calculator myCalcu = new Calculator();

        // 정사각형의 넓이 구하기
        double result1 = myCalcu.areaRectangle(10);

        // 직사각형의 넓이 구하기
        double result2 = myCalcu.areaRectangle(10, 20);

        // 결과 출력
        System.out.println("정사각형 넓이=" + result1);
        System.out.println("직사각형 넓이=" + result2);
    }
}
```