

6장. 클래스

Contents

- ❖ 8절. 메소드(Method)
- ❖ 9절. 인스턴스 멤버와 this
- ❖ 10절. 정적 멤버와 static
- ❖ 11절. final 필드와 상수(static final)

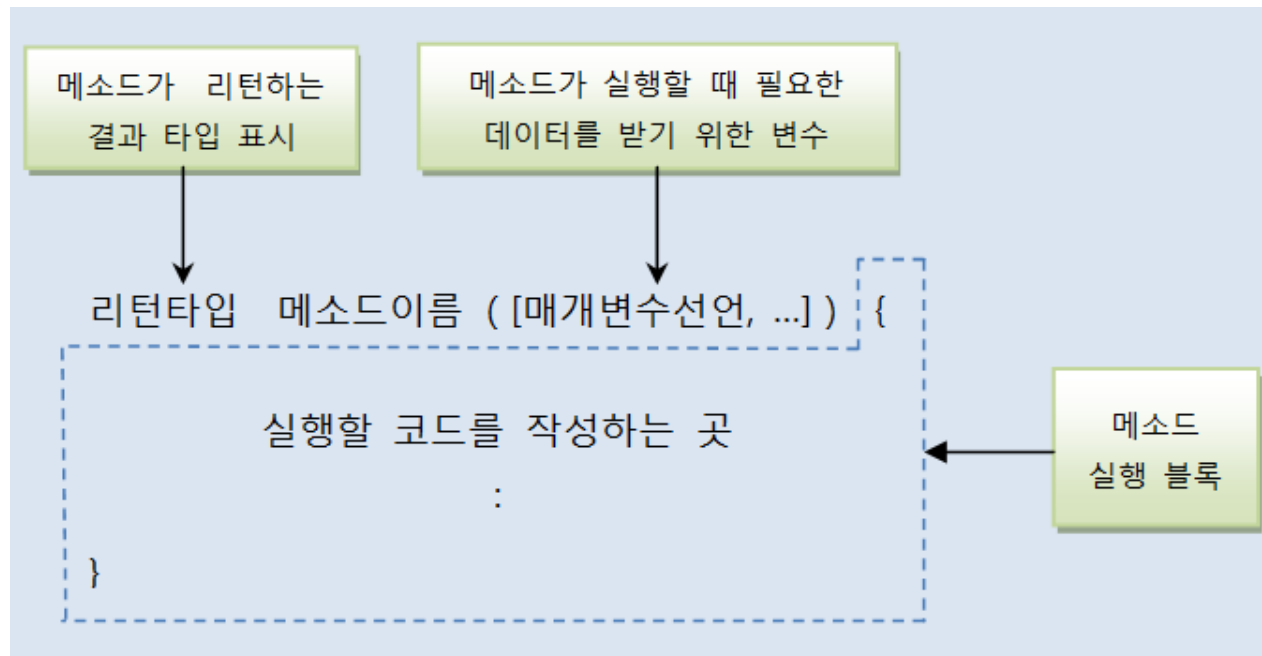
8절. 메소드(method)

❖ 메소드란?

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

- 필드를 읽고 수정하는 역할
- 다른 객체를 생성해서 다양한 기능을 수행
- 객체 간의 데이터 전달의 수단으로 사용

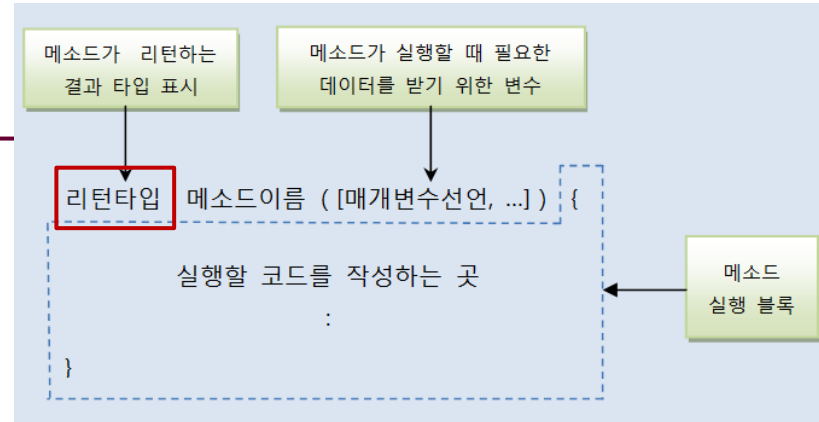
❖ 메소드 선언



8절. 메소드(method)

❖ 메소드 리턴 타입

- 메소드 실행된 후 리턴하는 값의 타입



- 메소드는 리턴값이 있을 수도 있고 없을 수도 있음

[메소드 선언]

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

[메소드 호출]

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

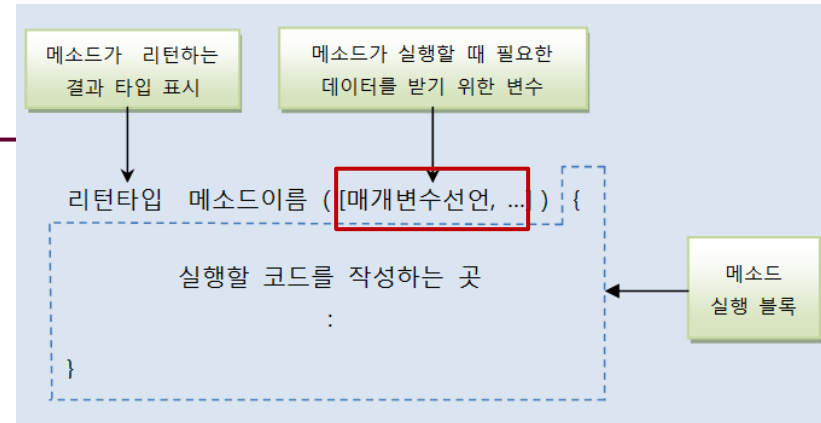
❖ 메소드 이름

- 자바 식별자 규칙에 맞게 작성
 - 숫자로 시작하면 안되고, \$와 _를 제외한 특수문자를 사용하지 말것
 - 관례적으로 메소드명은 소문자로 작성
 - 서로 다른 단어가 혼합된 이름이라면 뒤이어 오는 단어의 첫머리는 대문자로

8절. 메소드(method)

❖ 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);
```

8절. 메소드(method)

❖ 리턴(return) 문

- 메소드 실행을 중지하고 리턴값 지정하는 역할

- 리턴값이 있는 메소드

- 반드시 리턴(return)문 사용해 리턴값 지정해야

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

- return 문 뒤에 실행문 올 수 없음

- 리턴값이 없는 메소드

- 메소드 실행을 강제 종료 시키는 역할

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

8절. 메소드(method)

❖ Calculator 클래스

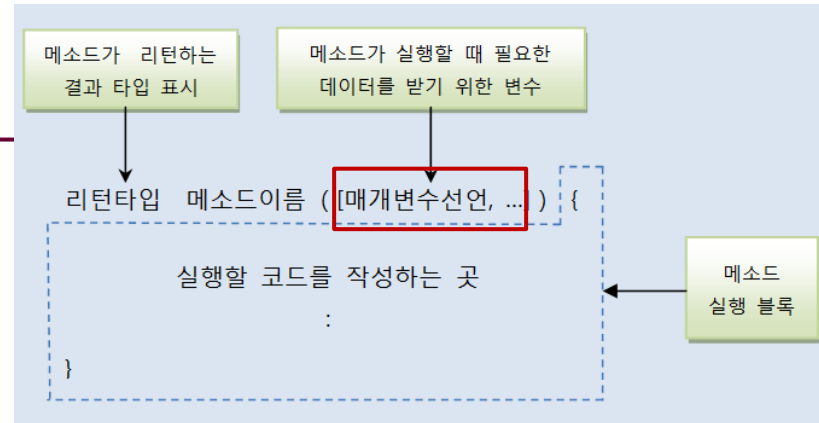
■ 속성

- 색상
- 크기

■ 생성자

■ 기능 (동작)

- 전원 켜기
- 전원 끄기
- 더하기
- 나누기



8절. 메소드(method)

❖ Calculator 클래스

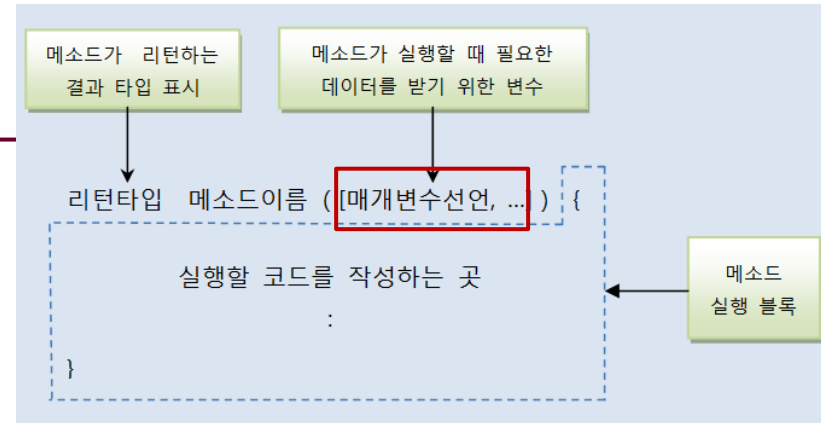
■ 속성

- 색상 : String color;
- 크기 : String size;

■ 생성자 : Calculator(String color, String size)

■ 기능 (동작)

- 전원 켜기 : void powerOn()
- 전원 끄기 : void powerOff()
- 더하기 : int plus(int x, int y)
- 나누기 : double divide(int x, int y)



8절. 메소드(method)

메소드가 리턴하는
결과 타입 표시

메소드가 실행할 때 필요한
데이터를 받기 위한 변수

❖ Calculator 클래스

■ 속성

- 색상 : String
- 크기 : String

■ 생성자 : Calcul

■ 기능 (동작)

- 전원 켜기 : void p
- 전원 끄기 : void p
- 더하기 : int plu
- 나누기 : double

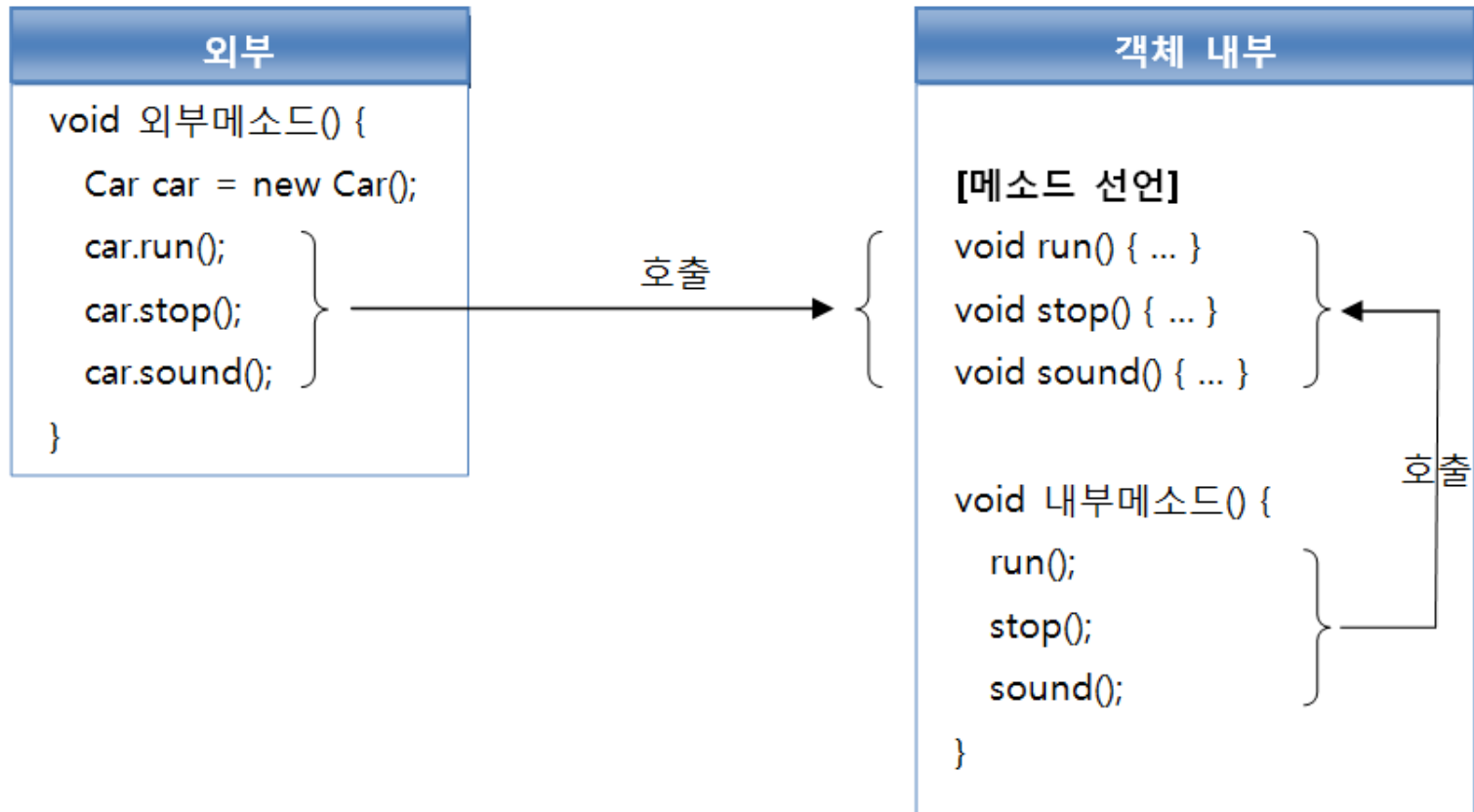
```
public class Calculator {  
    String color;  
    String size;  
  
    public Calculator(String color, String size) {  
        this.color = color;  
        this.size = size;  
    }  
    //전원켜기  
    //리턴값이 없는 경우  
    void powerOn() {  
        System.out.println("전원을 켭니다.");  
    }  
    //전원 끄기  
    //리턴값이 없는 경우  
    void powerOff() {  
        System.out.println("전원을 끕니다.");  
    }  
  
    int plus(int x, int y) {  
        return x+y;  
    }  
  
    double divide(int x, int y) {  
        return (double)x/y;  
    }  
}
```

메소드
실행 블록

8절. 메소드(method)

❖ 메소드 호출

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



8절. 메소드(method)

❖ 메소드 호출

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

```
public class CalcEx {  
    public static void main(String[] args) {  
        Calculator c1 = new Calculator("white", "small");  
        System.out.println(c1.color + ", " + c1.size);  
  
        int result1 = c1.plus(10, 20);  
        System.out.println("plus(10, 20) = " + result1);  
  
        double result2 = c1.divide(10, 20);  
        System.out.println("divide(10, 20) = " + result2);  
  
        byte b1 = 10;  
        byte b2 = 20;  
        result2 = c1.divide(b1, b2);  
        System.out.println("divide(10, 20) = " + result2);  
    }  
}
```

8절. 메소드(method)

❖ 메소드 호출

```
2 public class Calculator {
3     int plus(int x, int y) {
4         int result = x + y;
5         return result;
6     }
7     double avg(int x, int y) {
8         double sum = plus(x,y);
9         double result = sum / 2;
10        return result;
11    }
12    void exec() {
13        double result = avg(7,10);
14        printFunc("실행 결과 : " + result);
15    }
16    void printFunc(String strMsg) {
17        System.out.println(strMsg);
18    }
19 }
```

이용해 호출

```
class CalcEx {
    static void main(String[] args) {
        Calculator c1 = new Calculator("white", "small");
        System.out.println(c1.color + ", " + c1.size);

        int result1 = c1.plus(10, 20);
        System.out.println("plus(10, 20) = " + result1);

        double result2 = c1.divide(10, 20);
        System.out.println("divide(10, 20) = " + result2);

        byte b1 = 10;
        byte b2 = 20;
        result2 = c1.divide(b1, b2);
        System.out.println("divide(10, 20) = " + result2);

        c1.exec();
    }
}
```

8절. 메소드(method)

❖ 매개변수의 개수를 모를 경우

- 메소드의 매개변수는 개수가 정해져 있는 것이 일반적
- 경우에 따라서 메소드를 선언할 때 매개변수의 개수를 알 수 없는 경우 발생

[메소드 선언]

```
int sum1(int[] values) {    }
```

```
int sum2(int ... values) {    }
```

[메소드 호출]

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

```
int result1 = sum2(1, 2, 3);  
int result2 = sum2(1, 2, 3, 4, 5);
```

8절. 메소드(method)

❖ 매개변수의 개수를 모를 경우

Calculator.java

```
int sum1(int[] values) {  
    int sum = 0;  
    // for(int i : values)  
    //     sum += i;  
    for(int i=0; i<values.length; i++)  
        sum += values[i];  
    return sum;  
}  
  
int sum2(int...values) {  
    int sum = 0;  
    // for(int i : values)  
    //     sum += i;  
    for(int i=0; i<values.length; i++)  
        sum += values[i];  
    return sum;  
}
```

CalcEx.java

```
int sum1 = c1.sum1(new int[] {1,2,3,4,5});  
System.out.println("{1,2,3,4,5} = " + sum1);  
  
int sum2 = c1.sum2(1,2,3,4,5);  
System.out.println("(1,2,3,4,5) = " + sum2);
```

8절. 메소드(method)

❖ 메소드 오버로딩(Overloading)

- 클래스 내에 같은 이름의 메소드를 여러 개 선언하는 것
- 하나의 메소드 이름으로 다양한 매개값을 받기 위해 메소드 오버로딩
- 오버로딩의 조건: 매개변수의 타입, 개수, 순서 중 하나가 달라야 함

```
class 클래스 {  
    리턴타입 메소드이름 ( 타입 변수, ... ) { ... }  
}
```

리턴타입 메소드이름 (타입 변수, ...) { ... }

리턴타입 메소드이름 (타입 변수, ...) { ... }

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

plus(10, 20);

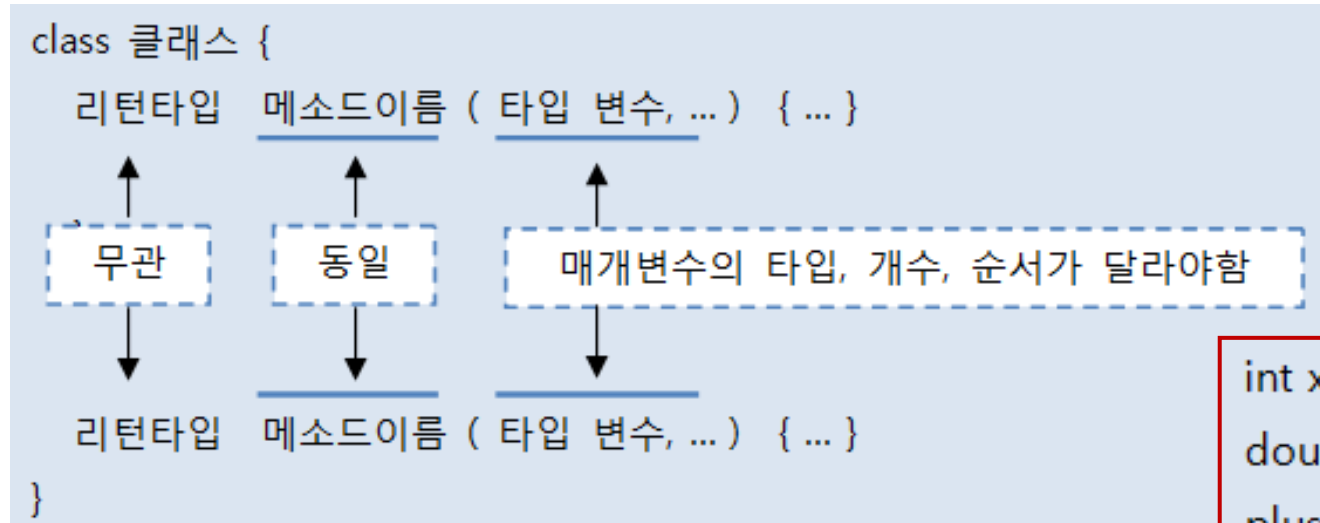
```
double plus(double x, double y) {  
    double result = x + y;  
    return result;  
}
```

plus(10.5, 20.3);

8절. 메소드(method)

❖ 메소드 오버로딩(Overloading)

- 클래스 내에 같은 이름의 메소드를 여러 개 선언하는 것
- 하나의 메소드 이름으로 다양한 매개값을 받기 위해 메소드 오버로딩
- 오버로딩의 조건: 매개변수의 타입, 개수, 순서 중 하나가 달라야 함



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

plus(10, 20);

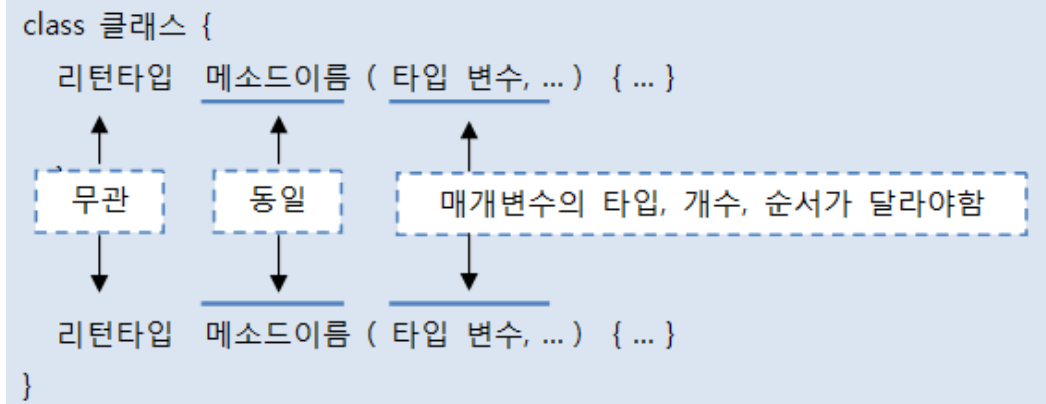
```
double plus(double x, double y) {  
    double result = x + y;  
    return result;  
}
```

plus(10.5, 20.3);

8절. 메소드(method)

❖ 메소드 오버로딩(Overloading)

- 클래스 내에 같은 이름의 메소드를 여러 개 선언하는 것
- 하나의 메소드 이름으로 다양한 매개값을 받기 위해 메소드 오버로딩
- 오버로딩의 조건: 매개변수의 타입, 개수, 순서 중 하나가 달라야 함



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



```
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) { .. }
```

8절. 메소드(method)

❖ 메소드 오버로딩(Overloading) 예제 (Area.java)

- 원, 정사각형, 직사각형의 넓이 구하기
 - areaCal() 메소드
- 원
 - 반지름을 입력받아 원의 면적을 계산해서 리턴(double 형식)
- 정사각형
 - 한 변의 길이를 입력받아 정사각형의 면적을 계산해서 리턴(int 형식)
- 직사각형
 - 두 변의 길이를 입력받아 직사각형의 면적을 계산해서 리턴(int 형식)

8절. 메소드(method)

❖ 메소드 오버로딩(Overloading)

■ 원, 정사각형, 직사각형의 넓이 구하기

- areaCal() 메소드

■ 원

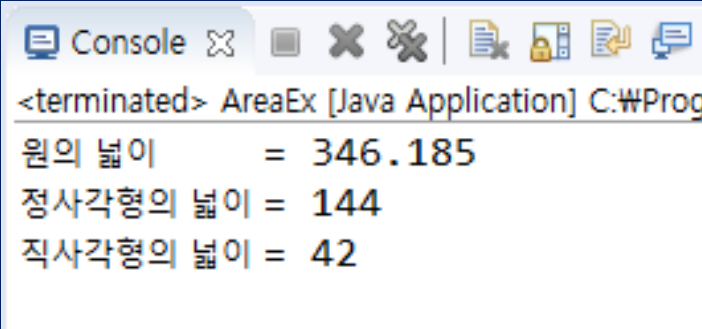
- 반지름을 입력받아 원의 면적을 계산해서 리턴(double 형식)

■ 정사각형

- 한 변의 길이를 입력받아 정사각형의 면적을 계산해서 리턴(int 형식)

■ 직사각형

- 두 변의 길이를 입력받아 직사각형의 면적을 계산해서 리턴(int 형식)



```
<terminated> AreaEx [Java Application] C:\#Prog
원의 넓이      = 346.185
정사각형의 넓이 = 144
직사각형의 넓이 = 42
```

8절. 메소드(method)

❖ 원, 정사각형, 직사각형의 넓이 구하기

```
2
3 public class Area {
4
5     double areaCal(double radius) {
6         return 3.14*radius*radius;
7     }
8
9     int areaCal(int width) {
10         return width*width;
11     }
12
13     int areaCal(int width, int height) {
14         return width*height;
15     }
16 }
```

8절. 메소드(method)

❖ 원, 정사각형, 직사각형의 넓이 구하기

```
2
3 public class Area {
4
5     double areaCal(double radius) {
6         return 3.14*radius*radius;
7     }
8
9     int
10
11 }
12
13     int
14
15 }
16 }
```

```
2
3 public class AreaEx {
4
5     public static void main(String[] args) {
6         Area areaObj = new Area();
7
8         double result1 = areaObj.areaCal(10.5);
9         System.out.println("원의 넓이 = " + result1);
10
11         int result2 = areaObj.areaCal(12);
12         System.out.println("정사각형의 넓이 = " + result2);
13
14         int result3 = areaObj.areaCal(6,7);
15         System.out.println("직사각형의 넓이 = " + result3);
16     }
17 }
```

8절. 메소드(method)

❖ 원, 정사각형, 직사각형의 넓이 구하기

```
2
3 public class Area {
4
5     double areaCal(double radius) {
6         return 3.14*radius*radius;
7     }
8
9     int
10
11 }
12
13 int
14
15 }
16 }
```

```
2
3 public class AreaEx {
4
5     public static void main(String[] args) {
6         Area areaObj = new Area();
7
8         double result1 = areaObj.areaCal(10.5);
9         System.out.println("원의 넓이 = " + result1);
10
11         int result2 = areaObj.areaCal(12);
12         System.out.println("정사각형의 넓이 = " + result2);
13
14         int result3 = areaObj.areaCal(6,7);
15         System.out.println("직사각형의 넓이 = " + result3);
16     }
17 }
```

Console [Java Application] C:\WProg
<terminated> AreaEx [Java Application] C:\WProg
원의 넓이 = 346.185
정사각형의 넓이 = 144
직사각형의 넓이 = 42

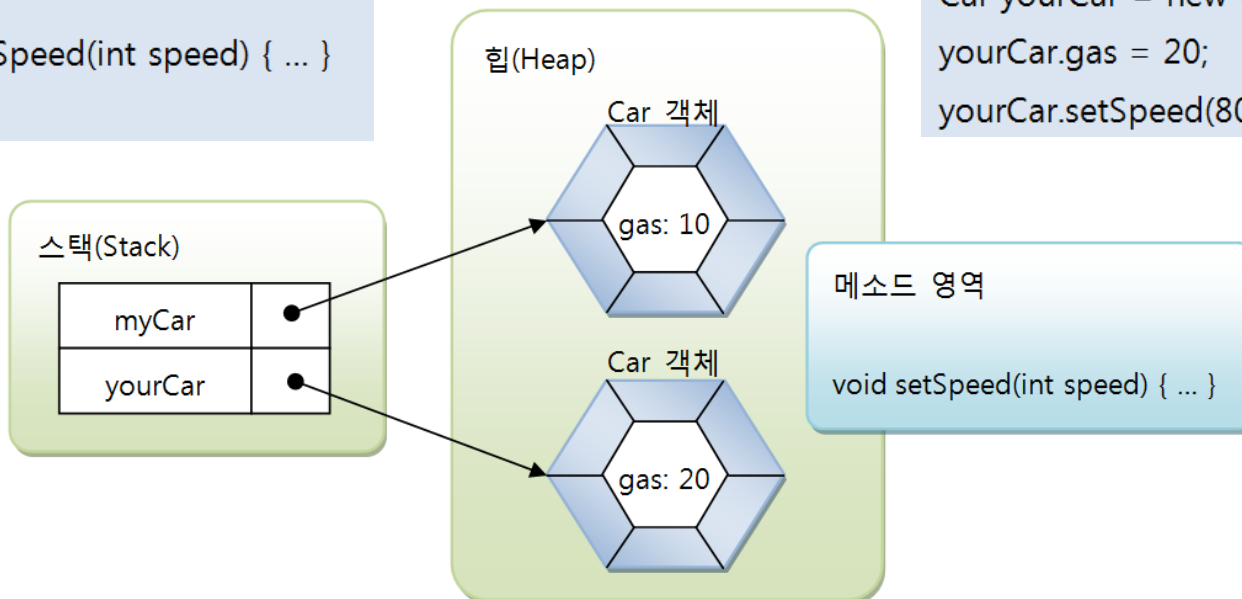
9절. 인스턴스 멤버와 this

❖ 인스턴스 멤버란?

- 객체(인스턴스)를 생성한 후 사용할 수 있는 필드와 메소드
 - 이들을 각각 인스턴스 필드, 인스턴스 메소드라고 부름
- 인스턴스 멤버는 객체에 소속된 멤버이기 때문에 객체가 없이 사용불가

```
public class Car {  
    //필드  
    int gas;  
  
    //메소드  
    void setSpeed(int speed) { ... }  
}
```

```
Car myCar = new Car();  
myCar.gas = 10;  
myCar.setSpeed(60);  
  
Car yourCar = new Car();  
yourCar.gas = 20;  
yourCar.setSpeed(80);
```



9절. 인스턴스 멤버와 this

❖ this

- 객체(인스턴스) 자신의 참조(번지)를 가지고 있는 키워드
- 객체 내부에서 인스턴스 멤버임을 명확히 하기 위해 this. 사용
- 매개변수와 필드명이 동일할 때 인스턴스 필드임을 명확히 하기 위해 사용

```
Car(String model) {  
    this.model = model;  
}  
  
void setModel(String model) {  
    this.model = model;  
}
```


10절. 정적 멤버와 static

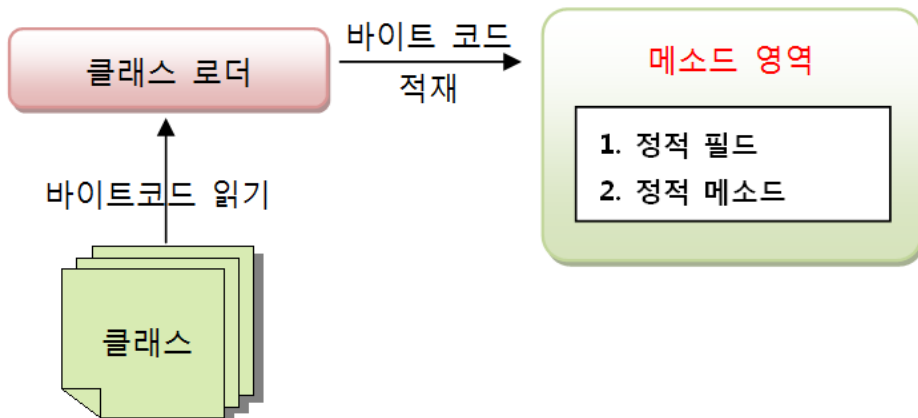
❖ 정적(static) 멤버란?

- 클래스에 고정된 필드와 메소드 - 정적 필드, 정적 메소드
- 정적 멤버는 클래스에 소속된 멤버
 - 객체 내부에 존재하지 않고, 메소드 영역에 존재
 - 정적 멤버는 객체를 생성하지 않고 클래스로 바로 접근해 사용

❖ 정적 멤버 선언

- 필드 또는 메소드 선언할 때 **static** 키워드 붙임

```
public class 클래스 {  
    //정적 필드  
    static 타입 필드 [= 초기값];  
  
    //정적 메소드  
    static 리턴타입 메소드( 매개변수선언, ... ) { ... }  
}
```



10절. 정적 멤버와 static

❖ 정적 멤버 사용

- 클래스 이름과 함께 도트(.) 연산자로 접근

```
클래스.필드;  
클래스.메소드( 매개값, ... );
```

```
public class Calculator {  
    static double pi = 3.14159;  
    static int plus(int x, int y) { ... }  
    static int minus(int x, int y) { ... }  
}
```

[바람직한 사용]

```
double result1 = 10 * 10 * Calculator.pi;  
int result2 = Calculator.plus(10, 5);  
int result3 = Calculator.minus(10, 5);
```

[바람직하지 못한 사용]

```
Calculator myCalcu = new Calculator();  
double result1 = 10 * 10 * myCalcu.pi;  
int result2 = myCalcu.plus(10, 5);  
int result3 = myCalcu.minus(10, 5);
```

10절. 정적 멤버와 static

❖ 정적 멤버 사용 (라이브러리 클래스)

```
Calculator.java CalculatorEx.java

package week9;
public class Calculator {
    static double pi = 3.14;
    static int plus(int x, int y) {
        return x+y;
    }
    static int minus(int x, int y) {
        return x-y;
    }
}
```

10절. 정적 멤버와 static

❖ 정적 멤버 사용 (실행 클래스)

```
Calculator.java  CalculatorEx.java ✖  
  
package week9;  
  
public class CalculatorEx {  
    public static void main(String[] args) {  
        //정적(static) 필드는 객체를 생성하지 않아도 접근이 가능하다  
        int radius = 8;  
        double circleArea = Calculator.pi * radius * radius;  
        System.out.printf("반지름이 %d인 원의 면적 = %.2f\n", radius, circleArea);  
  
        int x = 8;  
        int y = 7;  
        //정적(static) 메소드도 객체를 생성하지 않아도 접근이 가능하다  
        System.out.printf("(%d,%d)의 합 = %d\n", x, y, Calculator.plus(x, y));  
        System.out.printf("(%d,%d)의 차 = %d\n", x, y, Calculator.minus(x, y));  
    }  
}
```

10절. 정적 멤버와 static

❖ 정적 멤버 사용 (실행 클래스)

Calculator.java

CalculatorEx.java

```
package week9;

public class CalculatorEx {
    public static void main(String[] args) {
        //정적(static) 필드는 객체를 생성하지 않아도 접근이 가능하다
        int radius = 8;
        double circleArea = Calculator.pi * radius * radius;
        System.out.printf("반지름이 %d인 원의 면적 = %.2f\n", radius, circleArea);

        int x = 8;
        int y = 7;
        //정적(static) 메소드도 객체를 생성하지 않아도 접근이 가능하다
        System.out.printf("(%d,%d)의 합 = %d\n", x, y, Calculator.plus(x, y));
        System.out.printf("(%d,%d)의 차 = %d\n", x, y, Calculator.minus(x, y));
    }
}
```

Console

<terminated> CalculatorEx (1) [Java Application]

반지름이 8인 원의 면적 = 200.96

(8,7)의 합 = 15

(8,7)의 차 = 1

10절. 정적 멤버와 static

❖ 인스턴스 멤버 선언 vs 정적 멤버 선언의 기준

■ 필드

- 객체마다 가지고 있어야 할 데이터 → 인스턴스 필드
- 공용적인 데이터 → 정적 필드

```
public class Calculator {  
    String color;                //계산기 별로 색깔이 다를 수 있다.  
    static double pi = 3.14159; //계산기에서 사용하는 파이( $\pi$ )값은 동일하다.  
}
```

■ 메소드

- 인스턴스 필드로 작업해야 할 메소드 → 인스턴스 메소드
- 인스턴스 필드로 작업하지 않는 메소드 → 정적 메소드

```
public Calculator {  
    String color;  
    void setColor(String color) { this.color = color; }  
    static int plus(int x, int y) { return x + y; }  
    static int minus(int x, int y) { return x - y; }  
}
```

10절. 정적 멤버와 static

❖ 정적 초기화 블록

- 정적 필드는 객체 생성 없이도 사용해야 하므로 생성자에서 초기화 X
- 클래스가 메소드 영역으로 로딩될 때 자동으로 실행하는 블록

```
static {  
  
    ...  
  
}
```

- 정적 필드의 복잡한 초기화 작업과 정적 메소드 호출 가능
- 클래스 내부에 여러 개가 선언되면 선언된 순서대로 실행

10절. 정적 멤버와 static

❖ 정적 초기화 블록

- 정적 필드는 객체 생성 없이도 사용해야 하므로 생성자에서 초기화 X
- 클래스가 메소드 영역으로 로딩될 때 자동으로 실행하는 블록
- 정적 필드의 복잡한 초기화 작업과 정적 메소드 호출 가능
- 클래스 내부에 여러 개가 선언되면 선언된 순서대로 실행

```
public class Television {  
    static String company = "Samsung";  
    static String model = "LCD";  
    static String info;  
  
    static {  
        info = company + "-" + model;  
    }  
}
```


10절. 정적 멤버와 static

❖ 정적 메소드와 정적 블록 작성시 주의할 점

- 객체가 없어도 실행 가능
- 블록 내부에 인스턴스 필드나 인스턴스 메소드 사용 불가
- 객체 자신의 참조인 this 사용 불가
- 정적 메소드와 정적 블록에서 인스턴스 멤버를 사용하려면
 - 객체를 먼저 생성하고 참조변수로 접근 (예. main())

//인스턴스 필드와 메소드

```
int field1;  
void method1() { ... }
```

//정적 필드와 메소드

```
static int field2;  
static void method2() { ... }
```

//정적 블록

```
static {  
    field1 = 10;    (x)  
    method1();     (x)  
    field2 = 10;    (o)  
    method2();     (o)  
}
```

} 컴파일 에러

//정적 메소드

```
static void Method3 {  
    this.field1 = 10;    (x)  
    this.method1();     (x)  
    field2 = 10;        (o)  
    method2();          (o)  
}
```

} 컴파일 에러

```
static void Method3() {  
    ClassName obj = new ClassName();  
    obj.field1 = 10;  
    obj.method1();  
}
```

10절. 정적 멤버와 static

❖ 정적 메소드와 정적 블록 작성시 주의할 점

```
Car.java ✖
package week9;

public class Car {
    int speed;
    void run() {
        System.out.println("시속 " + speed + "km로 달립니다.");
    }

    //main() 메소드도 정적(static) 메소드이므로 객체 생성없이
    //인스턴스 필드나 인스턴스 메소드를 사용할 수 없다
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.speed = 100;
        myCar.run();
    }
}
```

10절. 정적 멤버와 static

❖ 정적 메소드와 정적 블록 작성시 주의할 점

```
Car.java ✖  
package week9;  
  
public class Car {  
    int speed;  
    void run() {  
        System.out.println("시속 " + speed + "km로 달립니다.");  
    }  
  
    //main() 메소드도 정적(static) 메소드이므로 객체 생성없이  
    //인스턴스 필드나 인스턴스 메소드를 사용할 수 없다  
    public static void main(String[] args) {  
        Car myCar = new Car();  
        myCar.speed = 100;  
        myCar.run();  
    }  
}
```

```
Console ✖  
<terminated> CarEx (1) [Java Application]  
시속 100km로 달립니다.
```

11절. final 필드와 상수(static final)

❖ final 필드

- 최종적인 값을 갖고 있는 필드 = 실행 도중 값을 변경할 수 없는 필드
- Final 필드의 딱 한번의 초기값 지정 방법
 - 필드 선언 시 초기값 대입
 - 생성자에서 초기값 대입

```
public class Person {  
    final String nation = "Korea";  
    final String ssn;  
    String name;  
  
    public Person(String ssn, String name) {  
        this.ssn = ssn;  
        this.name = name;  
    }  
}
```

11절. final 필드와 상수(static final)

❖ final 필드

- 최종적인 값을 갖고 있는 필드 = 실행 도중 값을 변경할 수 없는 필드
- final 필드의 딱 한번의 초기값 지정 방법
 - 필드 선언 시 초기값 대입
 - 생성자에서 초기값 대입

```
public class Person {  
    final String nation = "Korea";  
    final String ssn;  
    String name;  
  
    public Person(String ssn, String name) {  
        this.ssn = ssn;  
        this.name = name;  
    }  
}
```

고정된 값이라면 필드 선언 시
초기값 지정

매개변수로 받아서 초기화 한다면
생성자에서 초기값 지정

11절. final 필드와 상수(static final)

❖ 상수(static final)

- 상수 = 정적 final 필드
 - final 필드:
 - 객체마다 가지는 불변의 인스턴스 필드
 - 상수(static final):
 - 객체마다 가지고 있지 않음
 - 메소드 영역에 클래스 별로 관리되는 불변의 정적 필드
 - 공용 데이터로서 사용
- 상수 이름은 전부 대문자로 작성
- 다른 단어가 결합되면 _ 로 연결

11절. final 필드와 상수(static final)

❖ 예제

```
Person.java  PersonEx.java
package week9;

public class Person {
    static final double FEET_CONSTANT = 30.48;

    final String nation = "Korea";
    final String ssn;
    String name;
    double height;

    public Person(String ssn, String name, double height) {
        this.ssn = ssn;
        this.name = name;
        this.height = height;
    }
}
```

11절. final 필드와 상수(static final)

❖ 예제

```
Person.java  PersonEx.java ✖
package week9;

public class PersonEx {
    public static void main(String[] args) {
        Person p1 = new Person("123456-1234567", "홍길동", 6.1);

        System.out.println(p1.nation);
        System.out.println(p1.ssn);
        System.out.println(p1.name);
        System.out.println(p1.height*Person.FEET_CONSTANT + "cm");

        // final로 정의된 필드는 한 번 초기화된 후에는 값을 변경할 수 없다.
        //p1.nation = "USA";
        //p1.ssn     = "456789-2221111";
        p1.name     = "진달래";
    }
}
```


11절. final 필드와 상수(static final)

❖ 예제

Person.java PersonEx.java

```
package week9;

public class PersonEx {
    public static void main(String[] args) {
        Person p1 = new Person("123456-1234567", "홍길동", 6.1);

        System.out.println(p1.nation);
        System.out.println(p1.ssn);
        System.out.println(p1.name);
        System.out.println(p1.height*Person.FEET_CONSTANT + "cm");

        // final로 정의된 필드는 한 번 초기화된 후에는 값을 변경할 수 없다.
        //p1.nation = "USA";
        //p1.ssn     = "456789-2221111";
        p1.name     = "진달래";
    }
}
```

Console

<terminated> PersonEx [Java Application]

Korea
123456-1234567
홍길동
185.928cm

❖ 자동차 주행 프로그램(Car class)

■ 속성(필드)

- 연료

■ 기능(메소드)

- 연료 넣기
- 연료가 있는지 체크
 - . 연료 > 0 이면 return true
 - . 연료 == 0 이면 return false
- 달리기

❖ 실행 클래스

❖ 자동차 주행 프로그램(Car class)

■ 속성(필드)

- 연료

■ 기능(메소드)

- 연료 넣기
- 연료가 있는지 체크
 - . 연료 > 0 이면 return true
 - . 연료 == 0 이면 return false
- 달리기

❖ 실행 클래스

❖ 실행결과 → >>>>>>

```
gas가 있습니다.  
출발합니다.  
달립니다. (gas잔량:5)  
달립니다. (gas잔량:4)  
달립니다. (gas잔량:3)  
달립니다. (gas잔량:2)  
달립니다. (gas잔량:1)  
멈춥니다. (gas잔량:0)  
gas가 없습니다.
```

❖ 자동차 주행 프로그램(Car class)

■ 속성(필드)

- int gas

■ 기능(메소드)

- void setGas()
- boolean isLeftGas()
 - . gas > 0 이면 return true
 - . gas == 0 이면 return false
- void run()

❖ CarEx class

❖ 실행결과 → >>>>>>

```
gas가 있습니다.  
출발합니다.  
달립니다. (gas잔량:5)  
달립니다. (gas잔량:4)  
달립니다. (gas잔량:3)  
달립니다. (gas잔량:2)  
달립니다. (gas잔량:1)  
멈춥니다. (gas잔량:0)  
gas가 없습니다.
```

❖ Car class

- 필드 : int gas
- 메소드
 - void setGas()
 - boolean isLeftGas()
 - void run()

❖ CarEx class

❖ 실행결과 → >>>>>>

```
gas가 있습니다.  
출발합니다.  
달립니다. (gas잔량:5)  
달립니다. (gas잔량:4)  
달립니다. (gas잔량:3)  
달립니다. (gas잔량:2)  
달립니다. (gas잔량:1)  
멈춥니다. (gas잔량:0)  
gas가 없습니다.
```

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



```
public class Car {
    int gas;
    public Car() {
        System.out.println("Car 생성자 실행");
    }
    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;
    }

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

```
public class CarEx {
    public static void main(String[] args) {
        Car mycar = new Car();
        mycar.setGas(5);

        if (mycar.isLeftGas())
            System.out.println("출발합니다.");

        mycar.run();

        System.out.println("프로그램 종료");
    }
}
```

❖ Car class

- 필드 : int gas
- 메소드
 - void setGas()
 - boolean isLeftGas()
 - void run()

❖ CarEx class

- ❖ gas 입력은 Scanner 이용
- ❖ 0이 입력되면 프로그램 종료
- ❖ 실행결과 → >>>>>>

```
gas가 있습니다.  
출발합니다.  
달립니다. (gas잔량:5)  
달립니다. (gas잔량:4)  
달립니다. (gas잔량:3)  
달립니다. (gas잔량:2)  
달립니다. (gas잔량:1)  
멈춥니다. (gas잔량:0)  
gas가 없습니다.  
gas를 주입하세요 >> 3  
gas가 있습니다.  
출발합니다.  
달립니다. (gas잔량:3)  
달립니다. (gas잔량:2)  
달립니다. (gas잔량:1)  
멈춥니다. (gas잔량:0)  
gas가 없습니다.  
gas를 주입하세요 >> 0  
프로그램 종료
```

실습

❖ Car class

- 필드 : int gas
- 메소드
 - void setGas()
 - boolean isLeftGas()
 - void run()

❖ CarEx class

- ❖ gas 입력은 Scanner 이용
- ❖ 0이 입력되면 프로그램 종료
- ❖ 실행결과 → >>>>>>

```
public class Car2Ex {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int gasVal = 0;  
  
        Car mycar = new Car();  
        mycar.setGas(5);  
  
        boolean gasState = mycar.isLeftGas();  
  
        while(true) {  
            if(gasState) {  
                System.out.println("출발합니다.");  
                mycar.run();  
            }  
  
            System.out.print("gas를 넣으세요 >> ");  
            gasVal = sc.nextInt();  
            if (gasVal == 0)  
                break;  
            mycar.setGas(gasVal);  
            gasState = mycar.isLeftGas();  
        }  
        System.out.println("프로그램 종료");  
        sc.close();  
    }  
}
```