



## 6장. 클래스

# Contents

- ❖ 1절. 객체 지향 프로그래밍
- ❖ 2절. 객체(Object)와 클래스(Class)
- ❖ 3절. 클래스 선언
- ❖ 4절. 객체 생성과 클래스 변수
- ❖ 5절. 클래스의 구성 멤버
- ❖ 6절. 필드(Field)
- ❖ 7절. 생성자(Constructor)
- ❖ 8절. 메소드(Method)

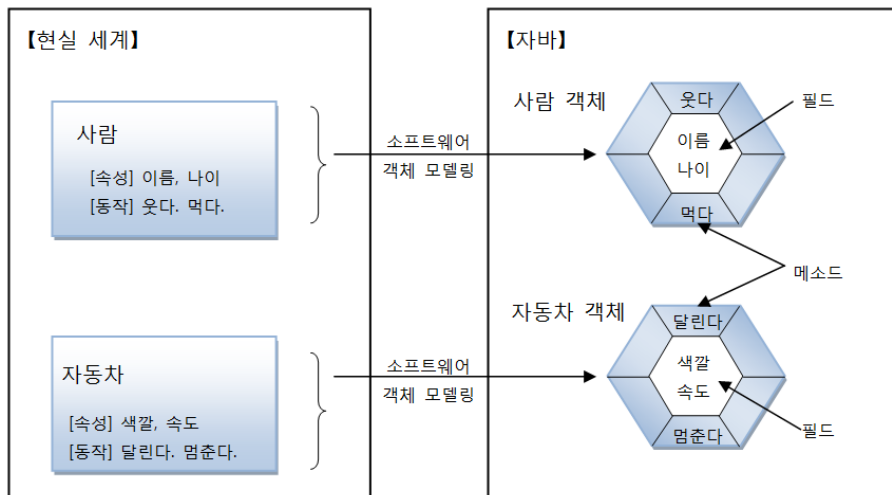
# 1절. 객체 지향 프로그래밍

## ❖ 객체 지향 프로그래밍

- OOP: Object Oriented Programming
- 부품 객체를 먼저 만들고 이것들을 하나씩 조립해 완성된 프로그램을 만드는 기법

## ❖ 객체(Object)란?

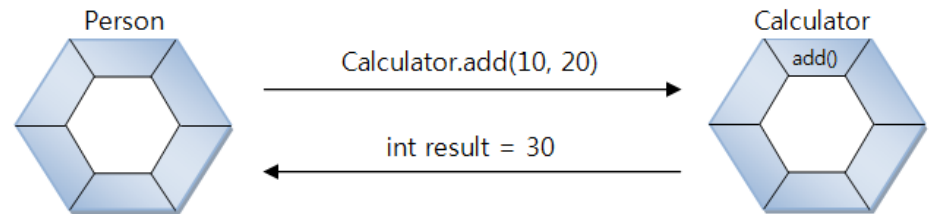
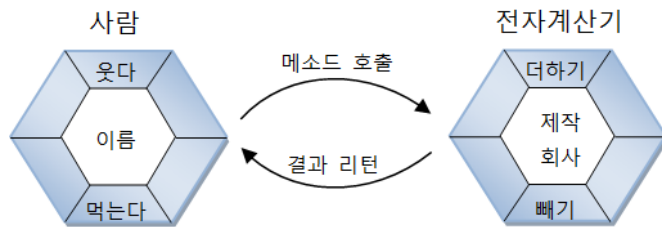
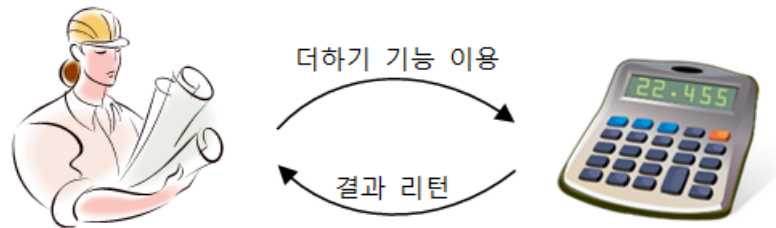
- 물리적으로 존재하는 것 (자동차, 책, 사람)
- 추상적인 것(회사, 날짜) 중에서 자신의 속성과 동작을 가지는 모든 것
- 객체는 필드(속성)와 메소드(동작)로 구성된 자바 객체로 모델링 가능



# 1절. 객체 지향 프로그래밍

## ❖ 객체의 상호 작용

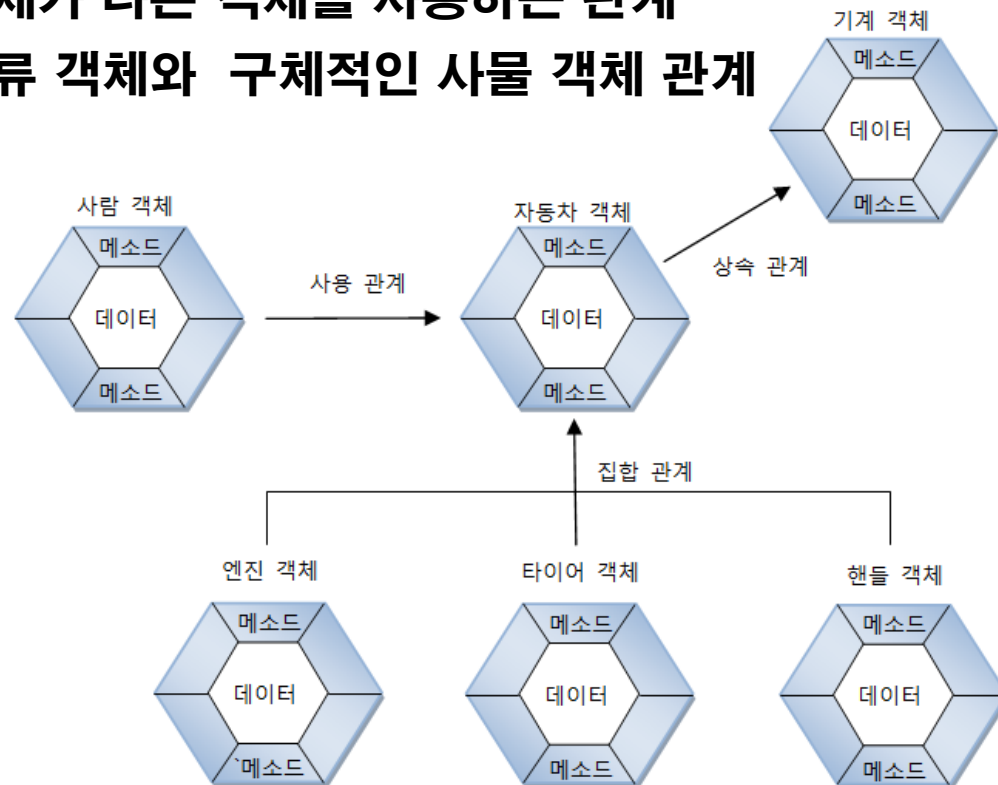
- 객체들은 서로 간에 기능(동작)을 이용하고 데이터를 주고 받음



# 1절. 객체 지향 프로그래밍

## ❖ 객체간의 관계

- 객체 지향 프로그램에서는 객체는 다른 객체와 관계를 맺음
- 관계의 종류
  - 집합 관계: 완제품과 부품의 관계
  - 사용 관계: 객체가 다른 객체를 사용하는 관계
  - 상속 관계: 종류 객체와 구체적인 사물 객체 관계

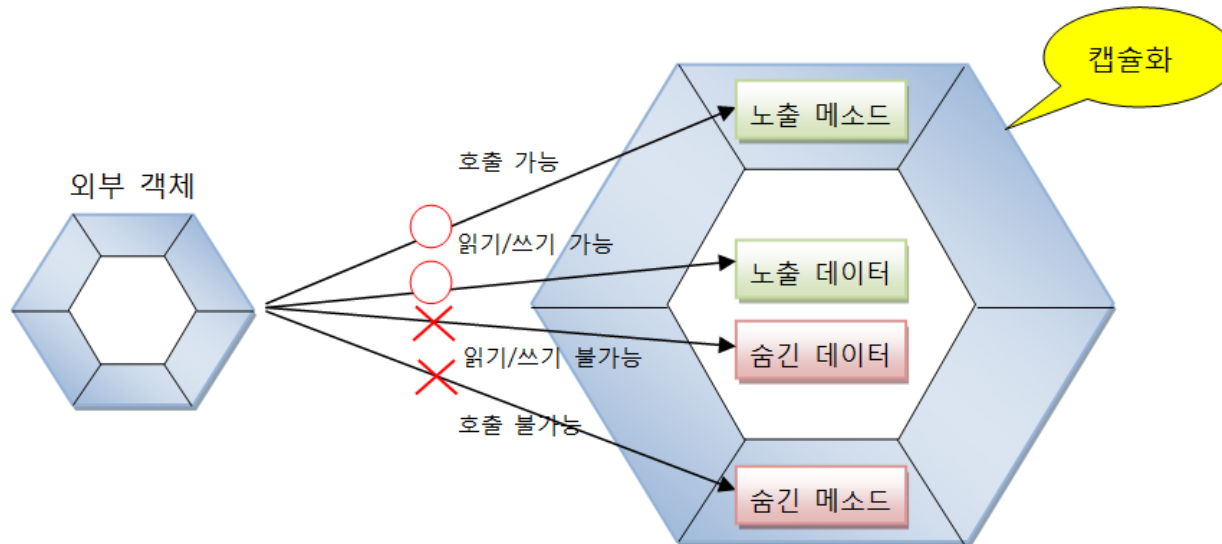


# 1절. 객체 지향 프로그래밍

## ❖ 객체 지향 프로그래밍의 특징

### ■ 캡슐화

- 객체의 필드, 메소드를 하나로 묶고, 실제 구현 내용을 감추는 것
- 외부 객체는 객체 내부 구조를 알지 못하며 객체가 노출해 제공하는 필드와 메소드만 이용 가능
- 필드와 메소드를 캡슐화하여 보호하는 이유는 외부의 잘못된 사용으로 인해 객체가 손상되지 않도록
- 자바 언어는 캡슐화된 멤버를 노출시킬 것인지 숨길 것인지 결정하기 위해 접근 제한자(Access Modifier) 사용

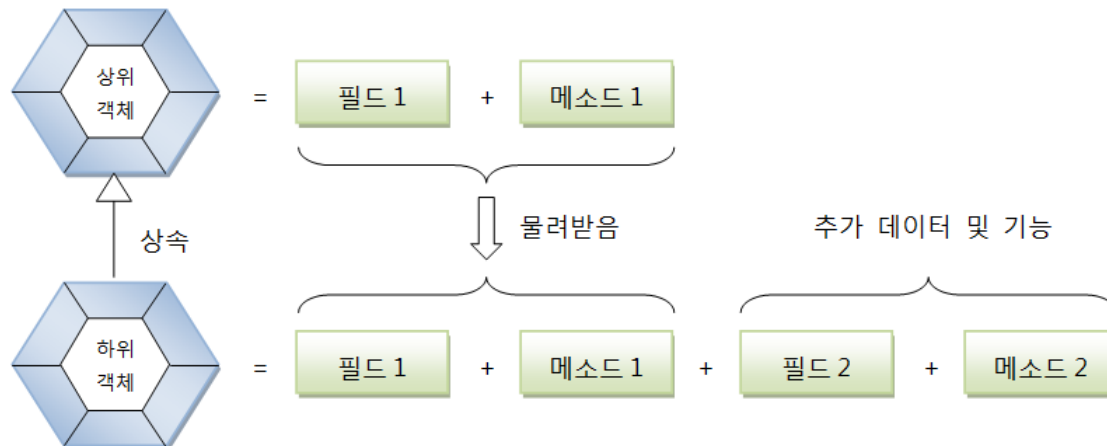


# 1절. 객체 지향 프로그래밍

## ❖ 객체 지향 프로그래밍의 특징

### ■ 상속

- 상위(부모) 객체의 필드와 메소드를 하위(자식) 객체에게 물려주는 행위
- 하위 객체는 상위 객체를 확장해서 추가적인 필드와 메소드를 가질 수 있음
- 상속 대상: 필드와 메소드
- 상속의 효과
  - 상위 객체를 재사용해서 하위 객체를 빨리 개발 가능
  - 반복된 코드의 중복을 줄임
  - 유지 보수의 편리성 제공
  - 객체의 다형성 구현



# 1절. 객체 지향 프로그래밍

## ❖ 객체 지향 프로그래밍의 특징

### ■ 다형성 (Polymorphism)

- 같은 타입이지만 실행 결과가 다양한 객체를 대입할 수 있는 성질
  - 부모 타입에는 모든 자식 객체가 대입
  - 인터페이스 타입에는 모든 구현 객체가 대입
- 효과
  - 객체를 부품화시키는 것 가능
  - 유지보수 용이

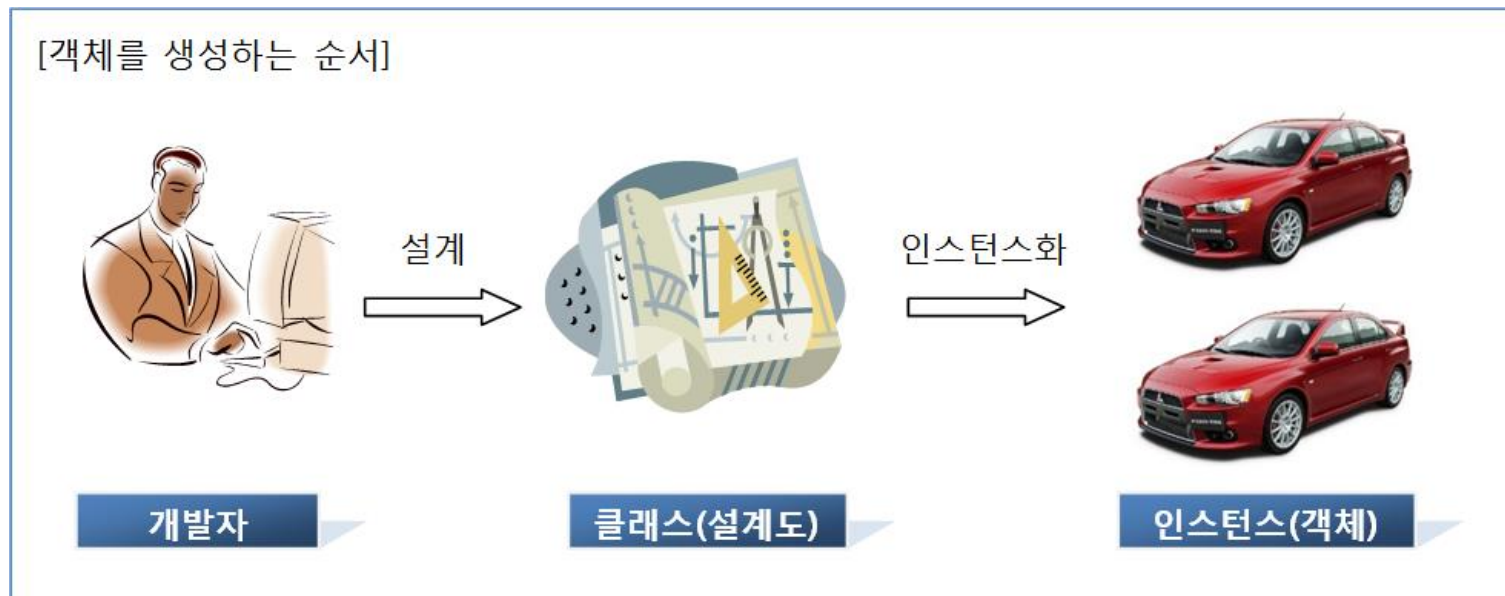




## 2절. 객체와 클래스

### ❖ 객체(Object)와 클래스(Class)

- 현실세계: 설계도 → 객체
- 자바: 클래스 → 객체
- 클래스에는 객체를 생성하기 위한 필드와 메소드가 정의
- 클래스로부터 만들어진 객체를 해당 클래스의 인스턴스(instance)
- 하나의 클래스로부터 여러 개의 인스턴스를 만들 수 있음



# 3절. 클래스 선언

## ❖ 클래스의 이름

### ■ 자바 식별자 작성 규칙에 따라야

번호	작성 규칙	예
1	하나 이상의 문자로 이루어져야 한다.	Car, SportsCar
2	첫 번째 글자는 숫자가 올 수 없다.	Car, 3Car(x)
3	'\$', '_' 외의 특수 문자는 사용할 수 없다.	\$Car, _Car, @Car(x), #Car(x)
4	자바 키워드는 사용할 수 없다.	int(x), for(x)

- 한글 이름도 가능하나, 영어 이름으로 작성
- 알파벳 대소문자는 서로 다른 문자로 인식
- 첫 글자와 연결된 다른 단어의 첫 글자는 대문자로 작성하는 것이 관례

Calculator, Car, Member, ChatClient, ChatServer, Web\_Browser

### 3절. 클래스 선언

#### ❖ 클래스 선언과 컴파일

- 소스 파일 생성: 클래스이름.java (대소문자 주의)
- 소스 작성

```
public class 클래스이름 {  
  
}
```

컴파일

javac.exe

클래스이름.class

- 소스 파일당 하나의 클래스를 선언하는 것이 관례
  - 두 개 이상의 클래스도 선언 가능
  - 소스 파일 이름과 동일한 클래스만 public으로 선언 가능
  - 선언한 개수만큼 바이트 코드 파일이 생성

Car.java

```
public class Car {  
  
}  
  
class Tire {  
  
}
```

컴파일

javac.exe

Car.class

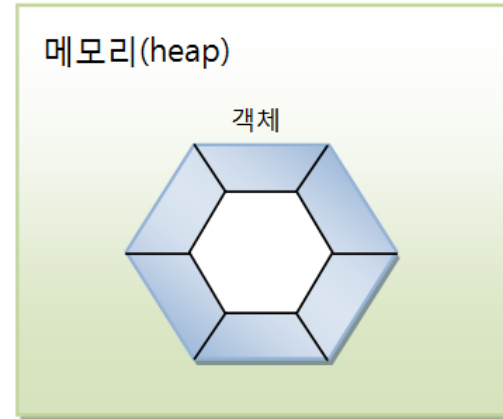
Tire.class

## 4절. 객체 생성과 클래스 변수

### ❖ new 연산자

#### ■ 객체 생성 역할

```
new 클래스();
```



- 클래스()는 생성자를 호출하는 코드
  - 생성된 객체는 힙 메모리 영역에 생성
- 
- new 연산자는 객체를 생성 후, 객체 생성 번지 리턴

## 4절. 객체 생성과 클래스 변수

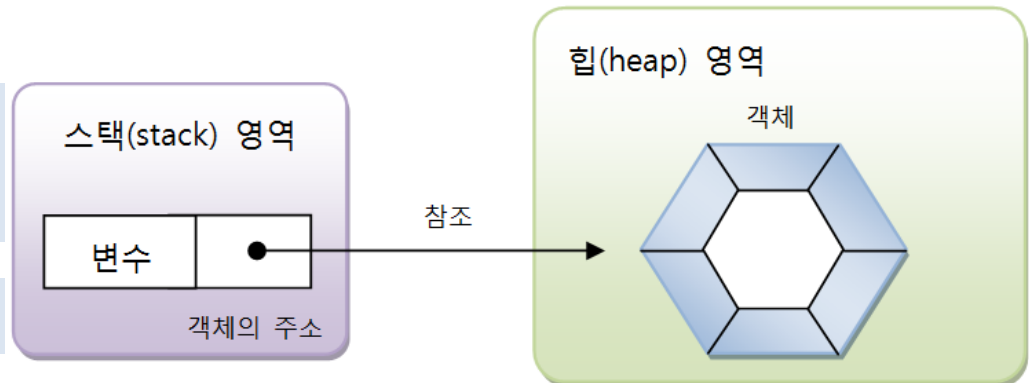
### ❖ 클래스 변수

- new 연산자에 의해 리턴 된 객체의 번지 저장 (참조 타입 변수)
- 힙 영역의 객체를 사용하기 위해 사용

클래스 변수;

변수 = new 클래스();

클래스 변수 = new 클래스();



## 4절. 객체 생성과 클래스 변수

### ❖ 클래스 변수

#### ■ 클래스 선언

```
public class Student {  
  
}
```

#### ■ 클래스로부터 객체 생성

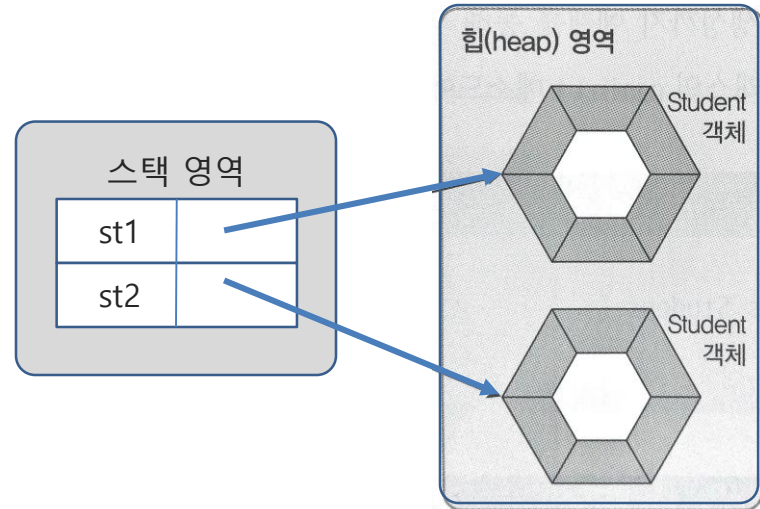
```
public class StudentEx {  
    public static void main(String[] args) {  
        Student st1 = new Student();  
        System.out.println("st1 변수가 Student 객체를 참조합니다");  
  
        Student st2 = new Student();  
        System.out.println("st2 변수가 또 다른 Student 객체를 참조합니다");  
    }  
}
```

## 4절. 객체 생성과 클래스 변수

### ❖ 클래스 변수

#### ■ 클래스 선언

```
public class Student {  
  
}
```



#### ■ 클래스로부터 객체 생성

```
public class StudentEx {  
    public static void main(String[] args) {  
        Student st1 = new Student();  
        System.out.println("st1 변수가 Student 객체를 참조합니다");  
  
        Student st2 = new Student();  
        System.out.println("st2 변수가 또 다른 Student 객체를 참조합니다");  
    }  
}
```

## 4절. 객체 생성과 클래스 변수

### ❖ 클래스의 용도

- 라이브러리(API: Application Program Interface) 용
  - 자체적으로 실행되지 않음
  - 다른 클래스에서 이용할 목적으로 만든 클래스
- 실행용
  - `main()` 메소드를 가지고 있는 클래스로 실행할 목적으로 만든 클래스

1개의 애플리케이션 = (1개의 실행클래스) + (n개의 라이브러리 클래스)



# 5절. 클래스의 구성 멤버

## ❖ 클래스의 구성 멤버

- 필드(Field)
- 생성자(Constructor)
- 메소드(Method)

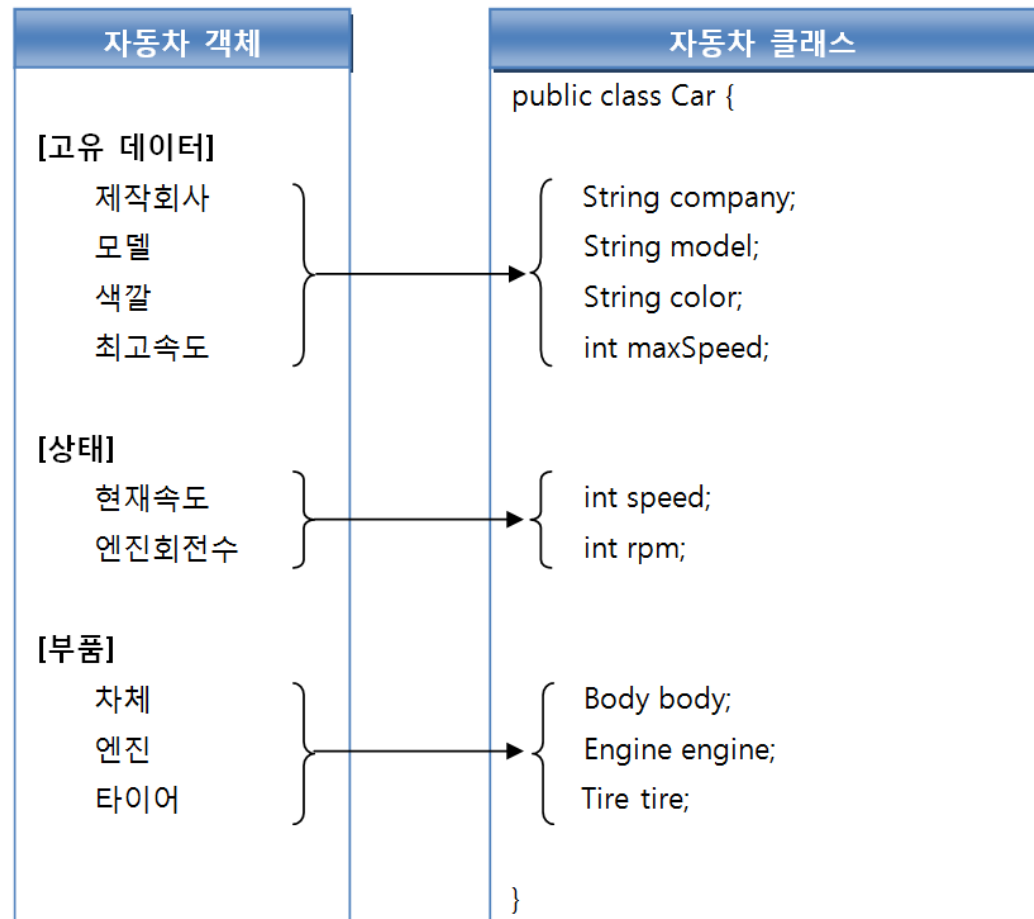
- 필드(Field) —————  
객체의 데이터가 저장되는 곳
- 생성자(Constructor) —————  
객체 생성시 초기화 역할 담당
- 메소드(Method) —————  
객체의 동작에 해당하는 실행 블록

```
public class ClassName {  
  
    //필드  
    int fieldName;  
  
    //생성자  
    ClassName() { ... }  
  
    //메소드  
    void methodName() { ... }  
  
}
```

## 6절. 필드(field)

### ❖ 필드의 내용

- 객체의 고유 데이터
- 객체가 가져야 할 부품 객체
- 객체의 현재 상태 데이터



## 6절. 필드(field)

### ❖ 필드 선언

타입 필드 [= 초기값] ;

```
String company = "현대자동차";  
String model = "그랜저";  
int maxSpeed = 300;  
int productionYear;  
int currentSpeed;  
boolean engineStart;
```

## 6절. 필드(field)

### ❖ 필드의 기본 초기값

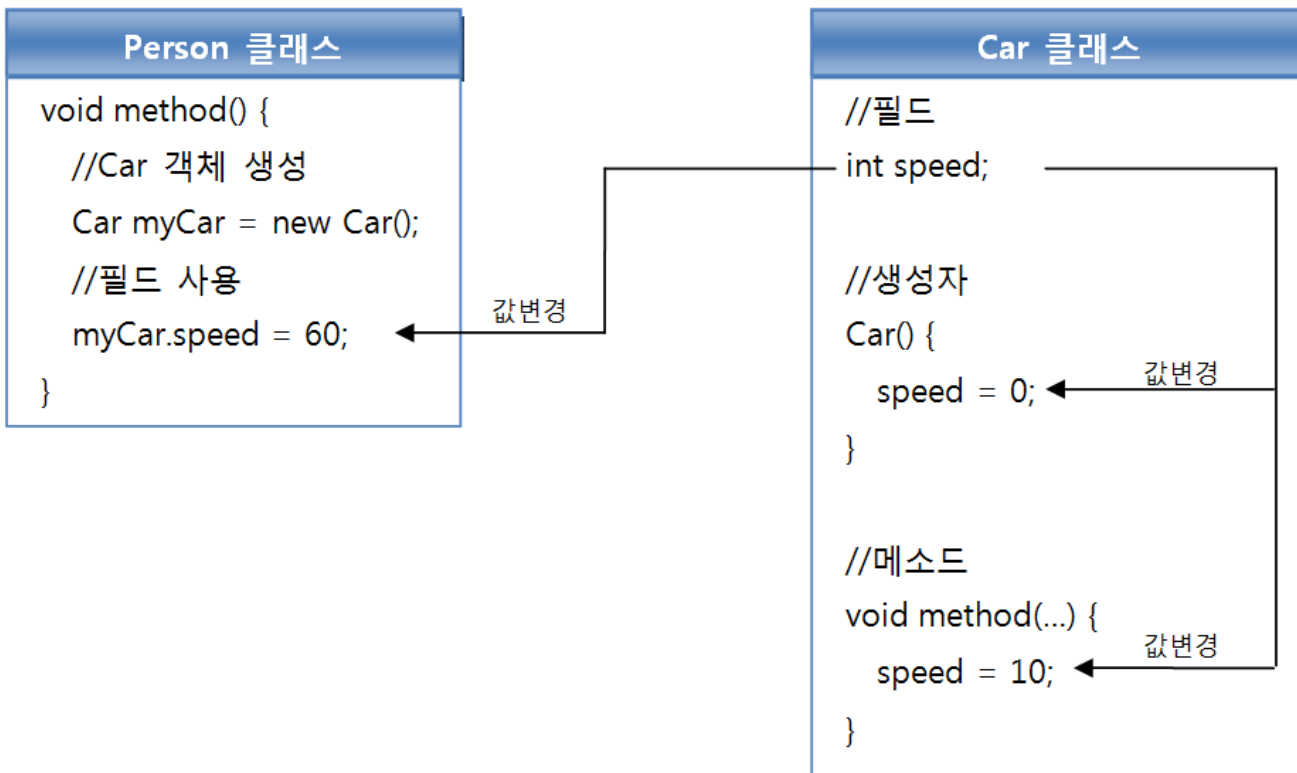
- 초기값 지정되지 않은 필드
  - 객체 생성시 자동으로 기본값으로 초기화

분류		데이터 타입	초기값
기본 타입	정수 타입	byte	0
		char	₩u0000 (빈 공백)
		short	0
		int	0
		long	0L
	실수 타입	float	0.0F
		double	0.0
	논리 타입	boolean	false
참조 타입		배열	null
		클래스(String 포함)	null
		인터페이스	null

## 6절. 필드(field)

### ❖ 필드 사용

- 필드 값을 읽고, 변경하는 작업을 말한다.
- 필드 사용 위치
  - 객체 내부: “**필드이름**” 으로 바로 접근
  - 객체 외부: “**변수.필드이름**” 으로 접근



## 6절. 필드(field)

### ❖ Car 클래스 필드 선언 및 외부 클래스에서 읽기

- 클래스 필드 내용
  - 객체의 고유 데이터
  - 객체가 가져야 할 부품 객체
  - 객체의 현재 상태 데이터

```
Car.java
1 package week7;
2
3 public class Car {
4     //Car 클래스 필드 선언
5     String company = "현대자동차";
6     String model   = "그랜저";
7     String color   = "검정";
8     int    maxSpeed = 350;
9     int    speed;
10 }
11
```

## 6절. 필드(field)

### ❖ Car 클래스 필드 선언 및 외부 클래스에서 읽기

- 외부 클래스에서 필드를 사용할 경우 반드시 클래스로부터 객체를 먼저 생성해야 한다
- 필드는 객체에 소속된 데이터이므로 객체가 존재하지 않으면 필드도 존재하지 않는다

```
CarEx.java
1 package week7;
2 public class CarEx {
3     public static void main(String[] args) {
4         //객체 생성
5         Car myCar = new Car();
6
7         //필드값 읽기
8         System.out.println("제작회사 : " + myCar.company);
9         System.out.println("모델명   : " + myCar.model);
10        System.out.println("색깔    : " + myCar.color);
11        System.out.println("최고속도 : " + myCar.maxSpeed);
12        System.out.println("현재속도 : " + myCar.speed);
13
14        //필드값 변경
15        myCar.speed = 60;
16        System.out.println("수정된 속도 : " + myCar.speed);
17    }
18 }
```

## 6절. 필드(field)

### ❖ Car 클래스 필드 선언 및 외부 클래스에서 읽기

- 외부 클래스에서 필드를 사용할 경우 반드시 클래스로부터 객체를 먼저 생성해야 한다
- 필드는 객체에 소속된 데이터이므로 객체가 존재하지 않으면 필드도 존재하지 않는다

```
CarEx.java
1 package week7;
2 public class CarEx {
3     public static void main(String[] args) {
4         //객체 생성
5         Car myCar = new Car();
6
7         //필드값 읽기
8         System.out.println("제작회사 : " + myCar.company);
9         System.out.println("모델명 : " + myCar.model);
10        System.out.println("색깔 : " + myCar.color);
11        System.out.println("최고속도 : " + myCar.maxSpeed);
12        System.out.println("현재속도 : " + myCar.speed);
13
14        //필드값 변경
15        myCar.speed = 60;
16        System.out.println("수정된 속도 : " + myCar.speed);
17    }
18 }
```

```
Console
<terminated> CarEx [Java Application] C:\Pr
제작회사 : 현대자동차
모델명 : 그랜저
색깔 : 검정
최고속도 : 350
현재속도 : 0
수정된 속도 : 60
```



# 7절. 생성자(Constructor)

## ❖ 생성자

- new 연산자에 의해 호출되어 객체의 초기화 담당

```
new 클래스();
```

- 필드의 값 설정
- 메소드 호출해 객체를 사용할 수 있도록 준비하는 역할 수행

## ❖ 기본 생성자(Default Constructor)

- 모든 클래스는 생성자가 반드시 존재하며 하나 이상 가질 수 있음
- 생성자 선언을 생략하면 컴파일러는 다음과 같은 기본 생성자 추가

```
[public] 클래스() { }
```

소스 파일(Car.java)

```
public class Car {  
  
}
```

→

바이트 코드 파일(Car.class)

```
public class Car {  
    public Car() { } //자동 추가  
}  
    기본 생성자
```

```
Car myCar = new Car();
```

기본 생성자

## 7절. 생성자(Constructor)

### ❖ 생성자 선언

- 디폴트 생성자 대신 개발자가 직접 선언

```
클래스( 매개변수선언, ... ) {  
    //객체의 초기화 코드  
}
```

} 생성자 블록

- 개발자 선언한 생성자 존재 시 컴파일러는 기본 생성자 추가하지 않음
  - new 연산자로 객체 생성시 개발자가 선언한 생성자 반드시 사용

```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

```
Car myCar = new Car("그랜저", "검정", 300);
```

# 7절. 생성자(Constructor)

생성자는 리턴 타입이 없고  
클래스 이름과 동일하다

## ❖ 생성자 선언

- 디폴트 생성자 대신 개발자가 직접 선언

```
클래스( 매개변수선언, ... ) {  
    //객체의 초기화 코드  
}
```

} 생성자 블록

- 개발자 선언한 생성자 존재 시 컴파일러는 기본 생성자 추가하지 않음
  - new 연산자로 객체 생성시 개발자가 선언한 생성자 반드시 사용

```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

```
Car myCar = new Car("그랜저", "검정", 300);
```

# 7절. 생성자(Constructor)

## ❖ 생성자 선언 및 객체 생성

### ■ 생성자 선언

```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

### ■ 생성자 호출을 통한 객체 생성

```
public class CarEx {  
    public static void main(String[] args) {  
        Car myCar = new Car("그랜저", "검정", 350);  
        //Car myCar = new Car();  
    }  
}
```

# 7절. 생성자(Constructor)

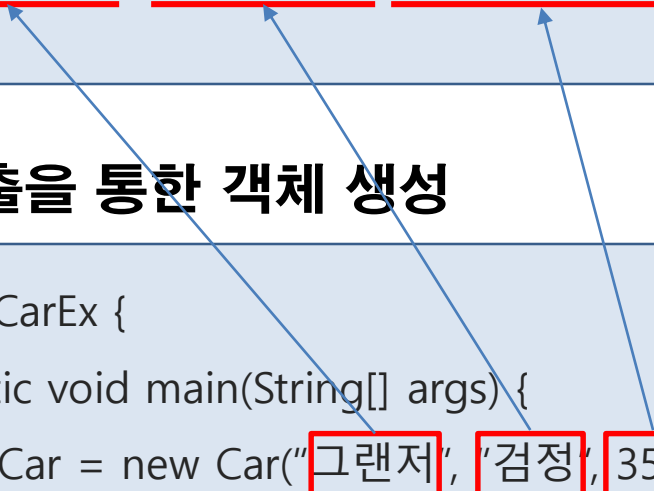
## ❖ 생성자 선언 및 객체 생성

### ■ 생성자 선언

```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

### ■ 생성자 호출을 통한 객체 생성

```
public class CarEx {  
    public static void main(String[] args) {  
        Car myCar = new Car("그랜저", "검정", 350);  
        //Car myCar = new Car();  
    }  
}
```



The diagram illustrates the flow of data from the CarEx class to the Car constructor. Three blue arrows originate from the arguments in the CarEx main method: the first arrow points from the string "그랜저" to the String model parameter in the Car constructor; the second arrow points from the string "검정" to the String color parameter; and the third arrow points from the integer 350 to the int maxSpeed parameter. The arguments "그랜저", "검정", and 350 are each enclosed in a red rectangular box.

# 7절. 생성자(Constructor)

## ❖ 생성자 선언 및 객체 생성

### ■ 생성자 선언

```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

### ■ 생성자 호출을 통한 객체 생성

```
public class CarEx {  
    public static void main(String[] args) {  
        Car myCar = new Car("그랜저", "검정", 350);  
        //Car myCar = new Car();  
    }  
}
```

기본 생성자를 호출할 수 없다

## 7절. 생성자(Constructor)

### ❖ 필드 초기화

- 초기값 없이 선언된 필드는 객체가 생성될 때 기본값으로 자동 설정
- 다른 값으로 필드 초기화하는 방법
  - 필드 선언할 때 초기값 설정
  - 생성자의 매개값으로 초기값 설정

```
Korean k1 = new Korean("박자바", "011225-1234567");  
Korean k2 = new Korean("김자바", "930525-0654321");
```

- 매개 변수와 필드명 같은 경우 this 사용

## 7절. 생성자(Constructor)

### ❖ 필드 초기화

- 초기값 없이 선언된 필드는 객체가 생성될 때 기본값으로 자동 설정
- 다른 값으로 필드 초기화하는 방법
  - 필드 선언할 때 초기값 설정

```
Car.java
1 package week7;
2
3 public class Car {
4     //Car 클래스 필드 선언
5     String company = "현대자동차";
6     String model   = "그랜저";
7     String color   = "검정";
8     int    maxSpeed = 350;
9     int    speed;
10 }
11
```



# 7절. 생성자(Constructor)

## ❖ 필드 초기화

- 초기값 없이 선언된 필드는 객체가 생성될 때 기본값으로 자동 설정
- 다른 값으로 필드 초기화하는 방법
  - 필드 선언할 때 초기값 설정
  - 생성자의 매개값으로 초기값 설정

```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

```
public class CarEx {  
    public static void main(String[] args) {  
        Car myCar = new Car("그랜저", "검정", 350);  
        //Car myCar = new Car();  
    }  
}
```

# 7절. 생성자(Constructor)

## ❖ 필드 초기화

- 초기값 없이 선언된 필드는 객체가 생성될 때 초기값이 자동으로 부여된다.
- 다른 값으로 필드 초기화하는 방법
  - 필드 선언할 때 초기값 설정
  - 생성자의 매개값으로 초기값 설정

```
public class Car {  
    //Car 클래스 필드 선언  
    String model;  
    String color;  
    int    maxSpeed;  
}
```

```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

```
public class CarEx {  
    public static void main(String[] args) {  
        Car myCar = new Car("그랜저", "검정", 350);  
        //Car myCar = new Car();  
    }  
}
```

# 7절. 생성자(Constructor)

## ❖ 필드 초기화

- 초기값 없이 선언된 필드는 객체가 생성될 때 초기값이 자동으로 설정된다.
- 다른 값으로 필드 초기화하는 방법
  - 필드 선언할 때 초기값 설정
  - 생성자의 매개값으로 초기값 설정

```
public class Car {  
    //Car 클래스 필드 선언  
    String model;  
    String color;  
    int    maxSpeed;  
}
```

```
public class Car {  
    //생성자  
    Car(String model, String color, int maxSpeed) { ... }  
}
```

```
public class CarEx {  
    public static void main(String[] args) {  
        Car myCar = new Car("그랜저", "검정", 350);  
        //Car myCar = new Car();  
    }  
}
```

필드명과  
매개변수가  
동일하다

# 7절. 생성자(Constructor)

## ❖ 필드 초기화

- 매개변수가 있는 생성자를 사용하는 경우

```
package week7;

public class Car {
    String model;
    String color;
    int    maxSpeed;

    Car(String m, String c, int maxSp){
        model    = m;
        color    = c;
        maxSpeed = maxSp;
    }
}
```

```
public class CarEx {
    public static void main(String[] args) {
        Car myCar = new Car("그랜저", "검정", 350);
        //Car myCar = new Car();
    }
}
```

# 7절. 생성자(Constructor)

## ❖ 필드 초기화

- 매개변수가 있는 생성자를 사용하는 경우

```
package week7;
```

```
public class Car {
```

```
    String model;
```

```
    String color;
```

```
    int maxSpeed;
```

```
    Car(String m, String c, int maxSp){
```

```
        model = m;
```

```
        color = c;
```

```
        maxSpeed = maxSp;
```

```
    }
```

```
}
```

필드명과 매개변수는 독립적이다

```
public class CarEx {
```

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

```
        Car myCar = new Car("그랜저", "검정", 350);
```

```
        //Car myCar = new Car();
```

```
}
```

# 7절. 생성자(Constructor)

## ❖ 필드 초기화

- 매개변수와 필드명이 같은 경우 **this**를 사용한다

```
package week7;

public class Car {
    String model;
    String color;
    int    maxSpeed;

    Car(String model, String color, int maxSpeed){
        this.model = model;
        this.color = color;
        this.maxSpeed = maxSpeed;
    }
}
```

객체 자신의 참조

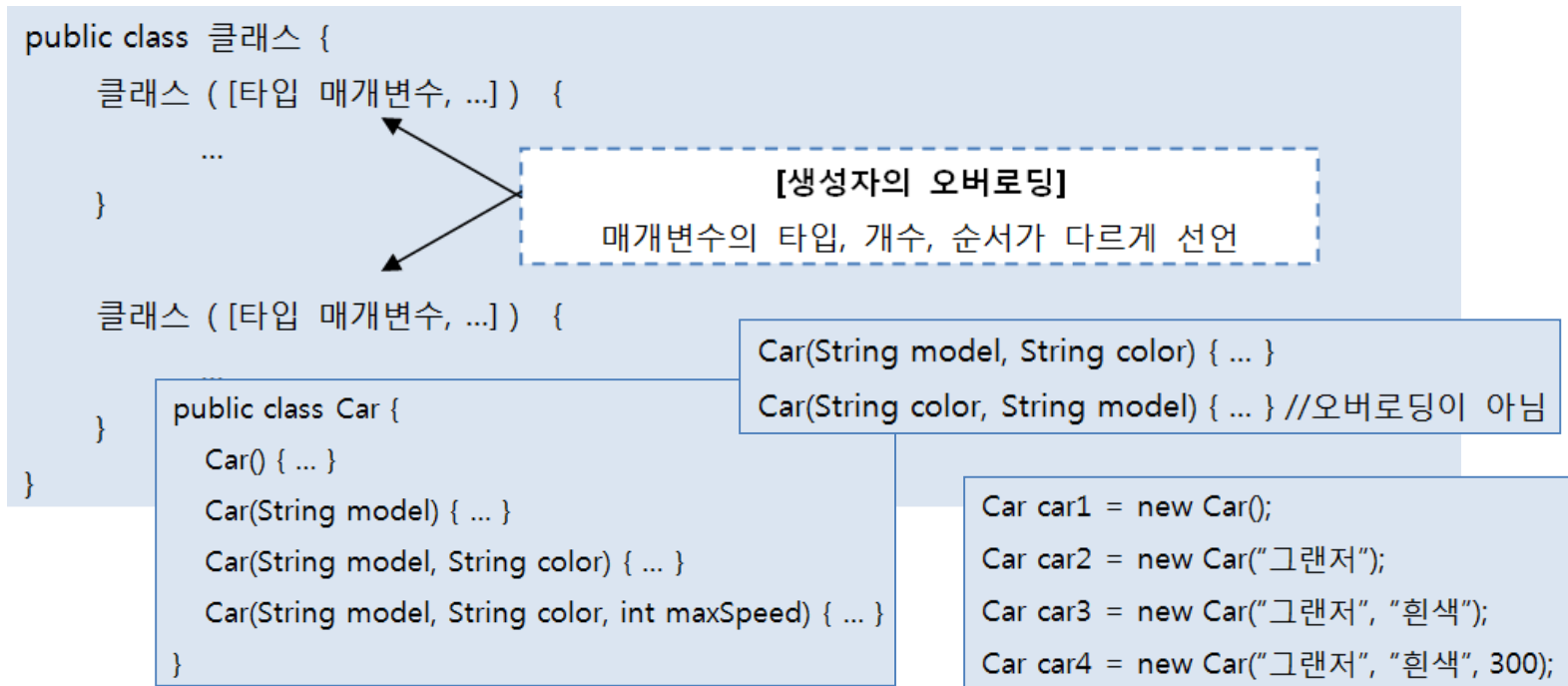
## 7절. 생성자(Constructor)

### ❖ 생성자를 다양화해야 하는 이유

- 객체 생성할 때 외부 값으로 객체를 초기화할 필요
- 외부 값이 어떤 타입으로 몇 개가 제공될 지 모름 - 생성자도 다양화

### ❖ 생성자 오버로딩(Overloading)

- 매개변수의 타입, 개수, 순서가 다른 생성자 여러 개 선언



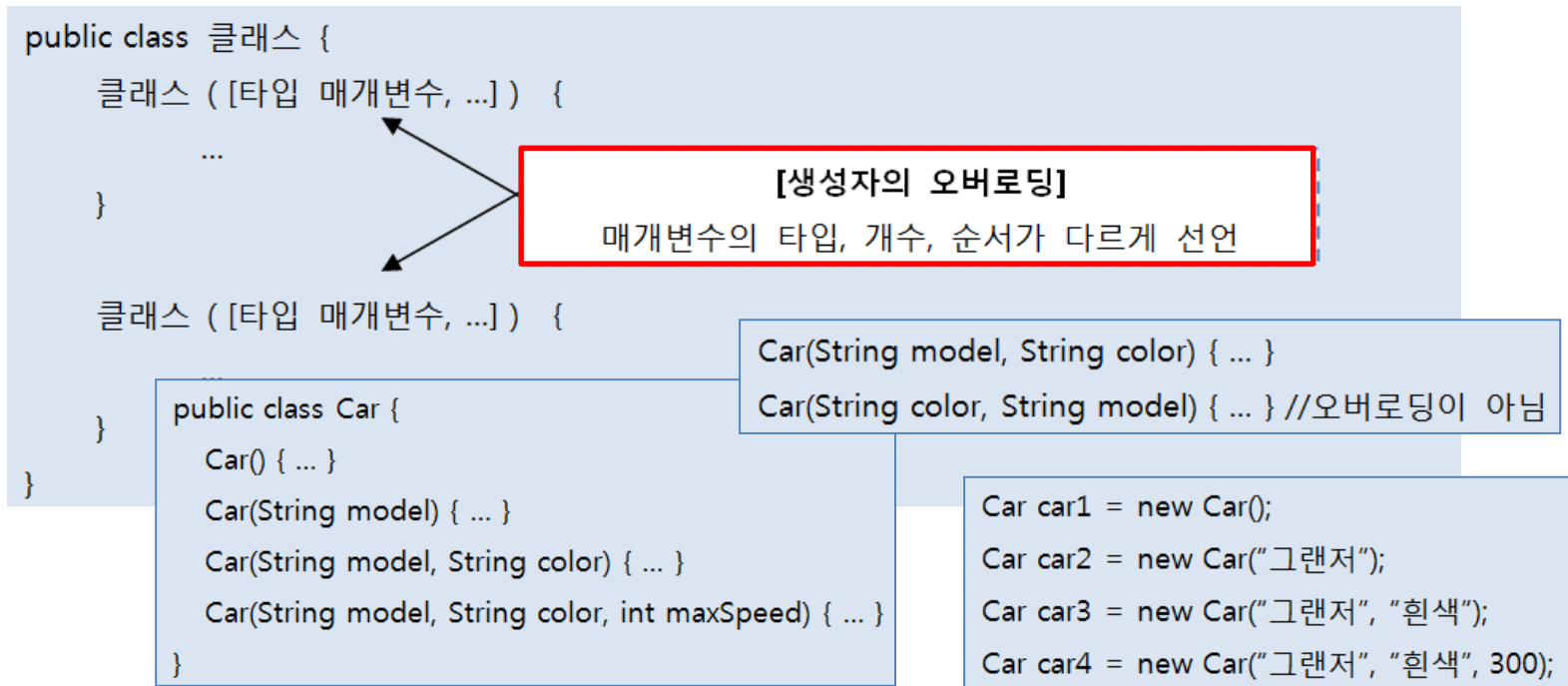
## 7절. 생성자(Constructor)

### ❖ 생성자를 다양화해야 하는 이유

- 객체 생성할 때 외부 값으로 객체를 초기화할 필요
- 외부 값이 어떤 타입으로 몇 개가 제공될 지 모름 - 생성자도 다양화

### ❖ 생성자 오버로딩(Overloading)

- 매개변수의 타입, 개수, 순서가 다른 생성자 여러 개 선언





## 7절. 생성자(Constructor)

### ❖ 다른 생성자 호출( this() )

- 생성자가 오버로딩되면 생성자 간의 중복된 코드 발생
- 초기화 내용이 비슷한 생성자들에서 이러한 현상을 많이 볼 수 있음
  - 초기화 내용을 한 생성자에 몰아 작성
  - 다른 생성자는 초기화 내용을 작성한 생성자를 this(...)로 호출

```
Car(String model) {  
    this.model = model;  
    this.color = "은색";  
    this.maxSpeed = 250;  
}
```

} 중복 코드

```
Car(String model, String color) {  
    this.model = model;  
    this.color = color;  
    this.maxSpeed = 250;  
}
```

} 중복 코드

```
Car(String model, String color, int maxSpeed) {  
    this.model = model;  
    this.color = color;  
    this.maxSpeed = maxSpeed;  
}
```

} 중복 코드

# 7절. 생성자(Constructor)

## ❖ 다른 생성자 호출( this() )

```
public class Car {
    String company = "현대자동차";
    String model;
    String color;
    int    maxSpeed;

    //생성자
    Car(){
    }

    Car(String model){
        this(model, "은색", 250);
    }

    Car(String model, String color){
        this(model, color, 250);
    }

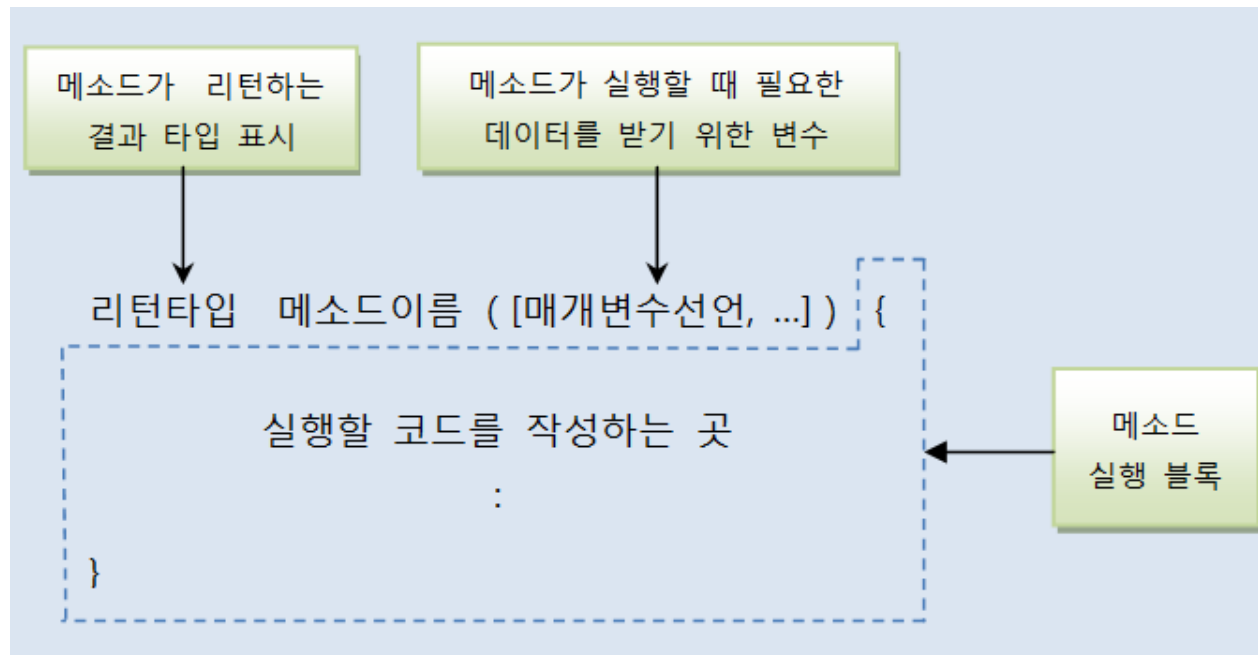
    Car(String model, String color, int maxSpeed){
        this.model    = model;
        this.color     = color;
        this.maxSpeed  = maxSpeed;
    }
}
```

## 8절. 메소드(method)

### ❖ 메소드란?

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

### ❖ 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

## 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;  
}
```

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

- return 문 뒤에 실행문 올 수 없음

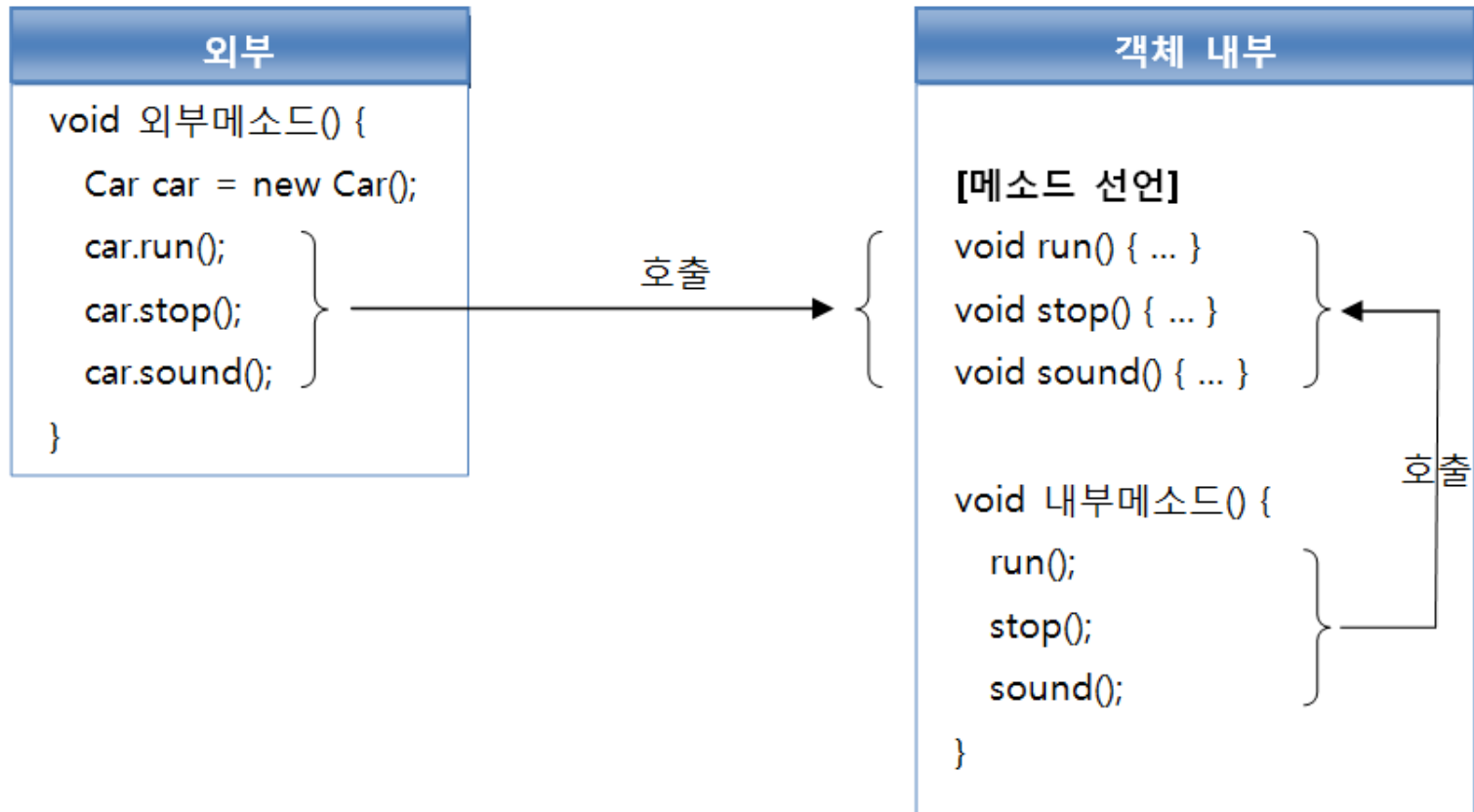
- 리턴값이 없는 메소드

- 메소드 실행을 강제 종료 시키는 역할

## 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

```
Calculator.java CalculatorEx.java
1 package week7;
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 }
```

```
Calculator.java CalculatorEx.java
1 package week7;
2
3 public class CalculatorEx {
4
5     public static void main(String[] args) {
6         Calculator myCal = new Calculator();
7         myCal.exec();
8     }
9 }
```



## 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

```
Calculator.java CalculatorEx.java
1 package week7;
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 }
```

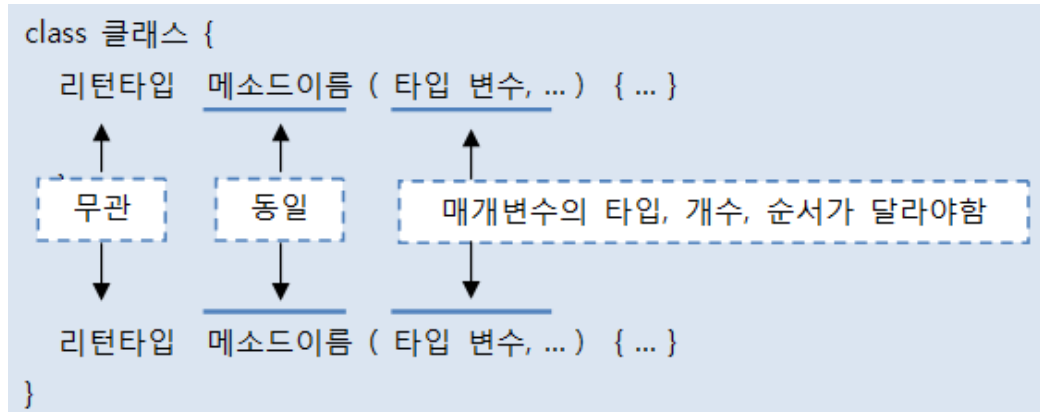
```
Calculator.java CalculatorEx.java
1 package week7;
2
3 public class CalculatorEx {
4
5     public static void main(String[] args) {
6         Calculator myCal = new Calculator();
7         myCal.exec();
8     }
9 }
```

```
Console
<terminated> CalculatorEx [Java Applica
실행 결과 : 8.5
```

## 8절. 메소드(method)

### ❖ 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, 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);
```

```
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) { .. }
```

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

?

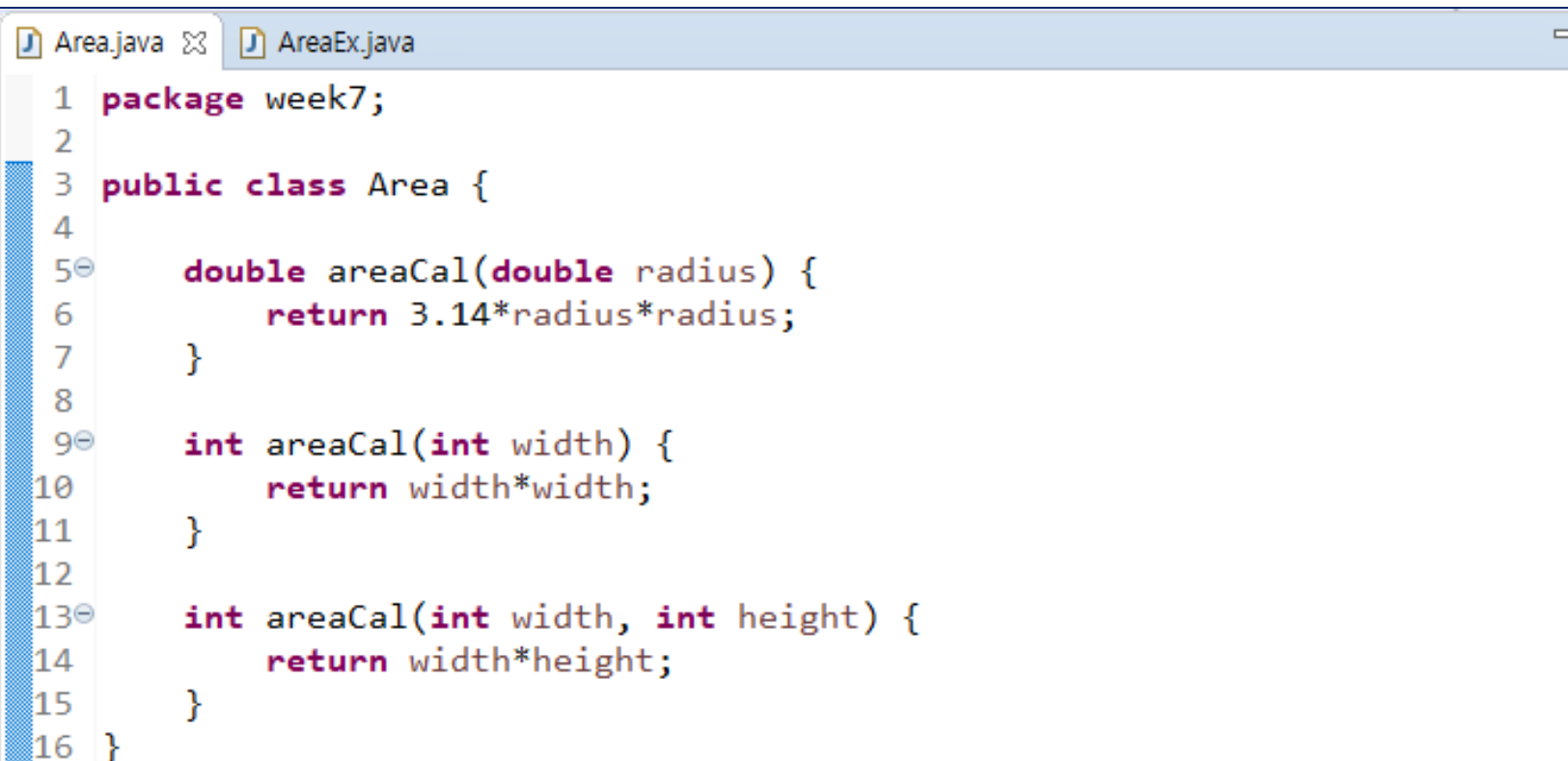
## 8절. 메소드(method)

### ❖ 메소드 오버로딩(Overloading) 예제

- 원, 정사각형, 직사각형의 넓이 구하기
  - 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)

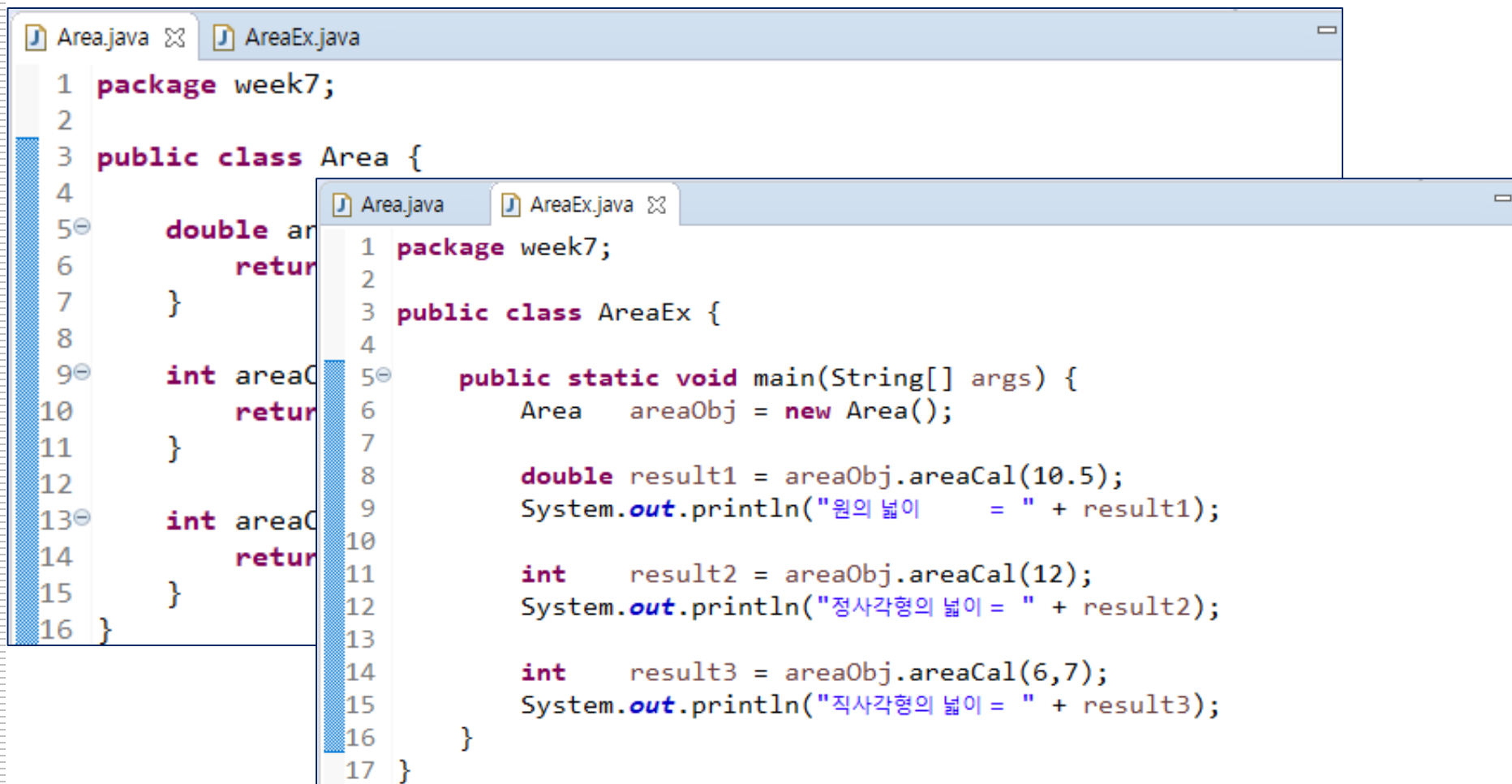
### ❖ 원, 정사각형, 직사각형의 넓이 구하기



```
1 package week7;
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

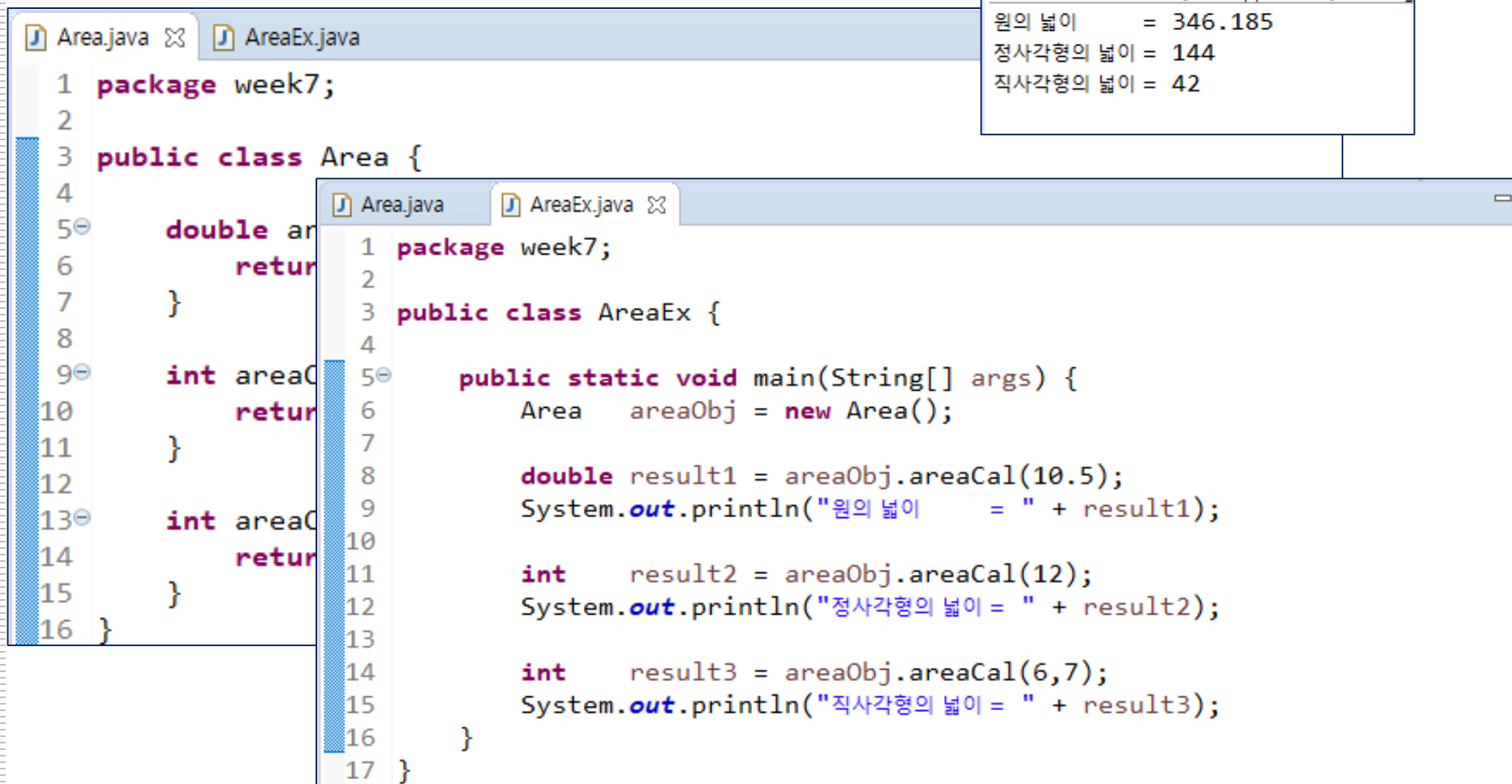


```
Area.java
1 package week7;
2
3 public class Area {
4
5     double areaCal(double radius) {
6         return Math.PI * radius * radius;
7     }
8
9     int areaCal(int side) {
10        return side * side;
11    }
12
13    int areaCal(int width, int height) {
14        return width * height;
15    }
16 }

AreaEx.java
1 package week7;
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



```
Area.java
1 package week7;
2
3 public class Area {
4
5     double areaCal(double radius) {
6         return Math.PI * radius * radius;
7     }
8
9     int areaCal(int side) {
10        return side * side;
11    }
12
13    int areaCal(int width, int height) {
14        return width * height;
15    }
16 }

AreaEx.java
1 package week7;
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

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