

Chapter

06

클래스

06-1. 객체 지향 프로그래밍

❖ 목차

- 시작하기 전에
- 객체의 상호작용
- 객체 간의 관계
- 객체와 클래스
- 클래스 선언
- 객체 생성과 클래스 변수
- 클래스의 구성 멤버
- 키워드로 끝내는 핵심 포인트

시작하기 전에

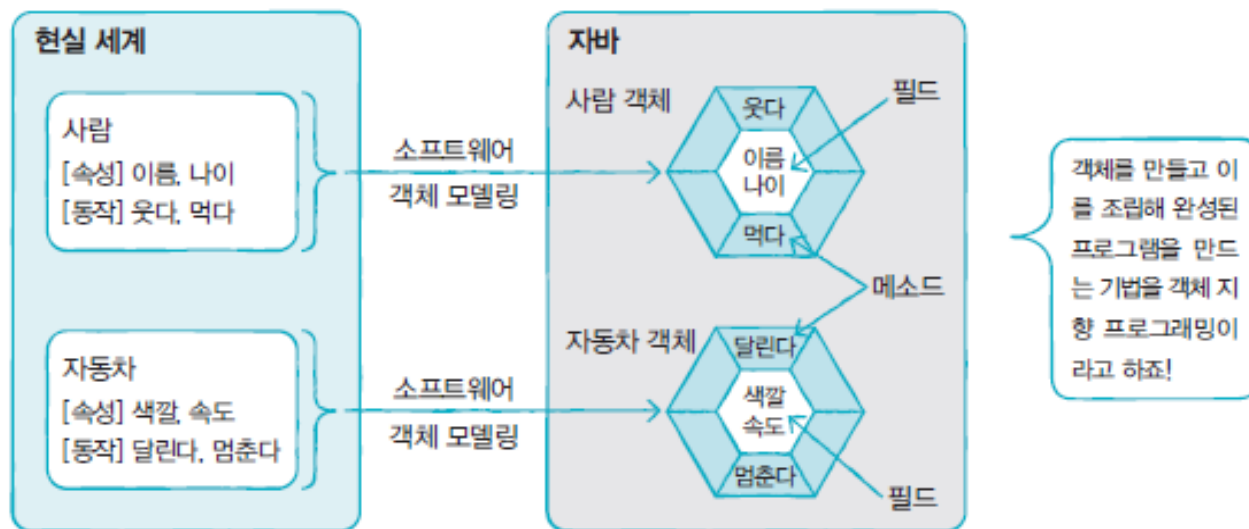
[핵심 키워드] : 클래스, 객체, new 연산자, 클래스 변수, 인스턴스, 클래스 멤버

[핵심 포인트]

객체의 개념과 객체의 상호작용에 대해 알아본다.
클래스로부터 객체를 생성하고 변수로 참조한다.

객체 (Object)

- 물리적으로 존재하거나 추상적으로 생각할 수 있는 것 중에서 자신의 속성을 가지며 식별 가능한 것
- 속성 (필드(field)) + 동작(메소드(method))로 구성



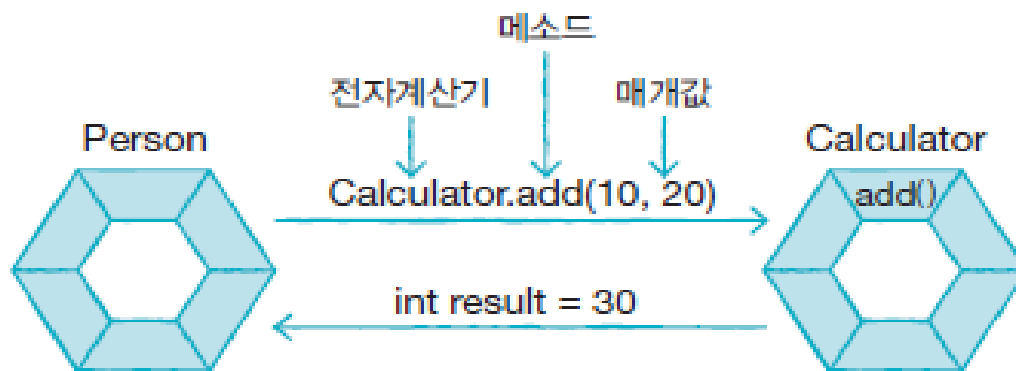
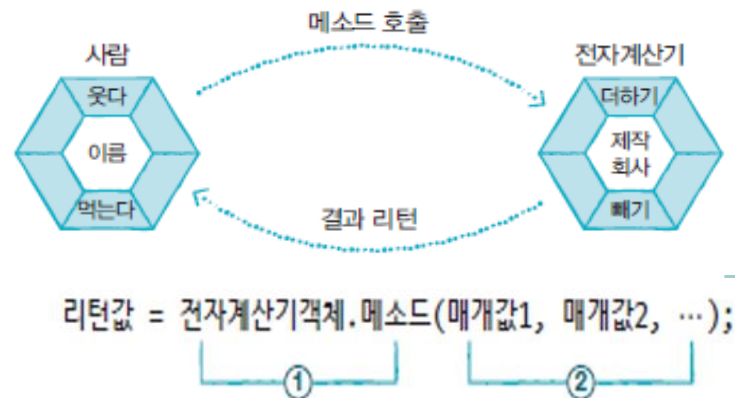
객체의 상호작용

❖ 객체와 객체 간의 상호작용

- 메소드를 통해 객체들이 상호작용
- **메소드 호출** : 객체가 다른 객체의 기능을 이용하는 것

```
int result = Calculator.add(10, 20);
```

↑
리턴한 값을 int 변수에 저장



객체 간의 관계

❖ 객체 간의 관계

■ 집합 관계

- 부품과 완성품의 관계
- 예: 부품과 자동차 관계

■ 사용 관계

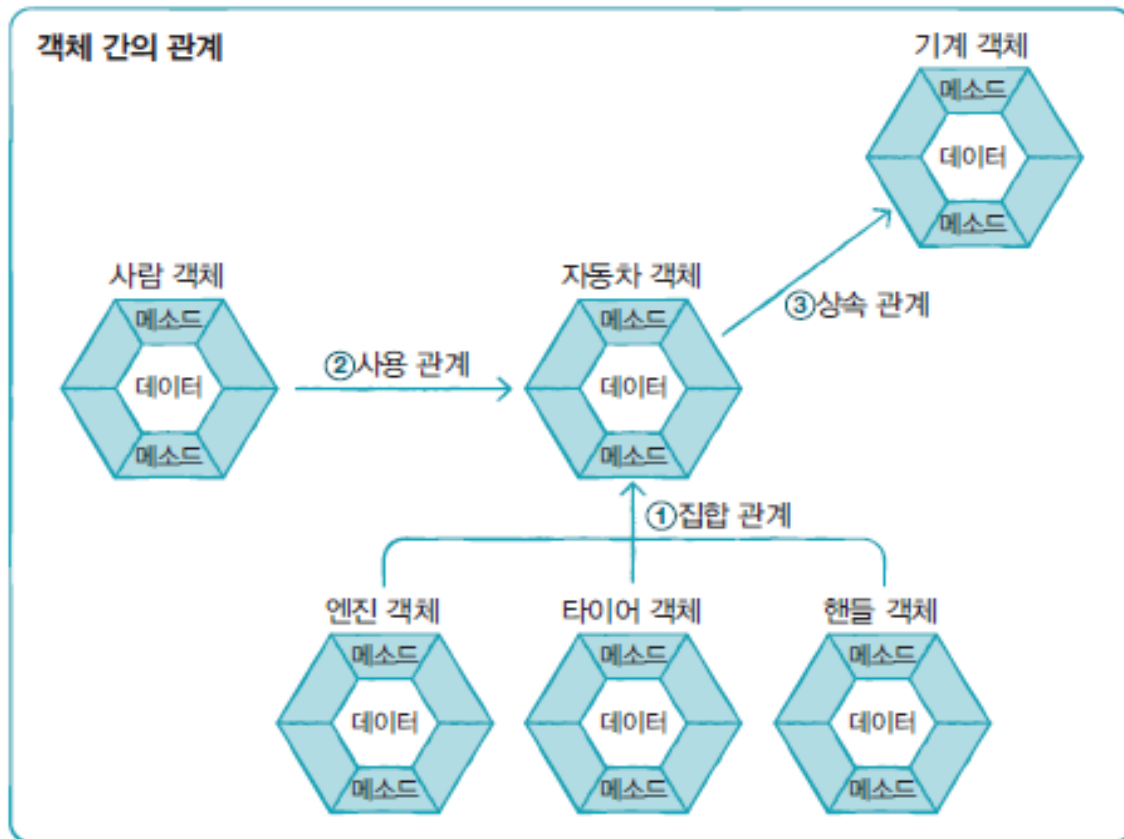
- 객체 간의 상호작용
- 예: 사람과 자동차 관계

■ 상속 관계

- 상위(부모) 객체를 기반으로
- 하위(자식) 객체를 생성
- 예: 기계와 자동차

■ 객체 지향 프로그래밍

- 집합/사용 관계에 있는 객체를 하나씩 설계한 후 조립하여 프로그램 개발



객체와 클래스

❖ 클래스 (class)

- 자바의 설계도
- **인스턴스** (instance): 클래스로부터 만들어진 객체
- 객체지향 프로그래밍 단계
 - 클래스 설계 -> 설계된 클래스로 사용할 객체 생성 -> 객체 이용

객체를 생성하는 순서



설계도는 클래스,
클래스로 만든 객체는 인스턴스

클래스 선언

❖ 클래스 선언

- 객체 구상 후 클래스 이름을 결정
 - 식별자 작성 규칙에 따라야 함
 - 하나 이상의 문자로 이루어질 것
 - 첫 글자에는 숫자 올 수 없음
 - \$, _ 외의 특수 문자는 사용할 수 없음
 - 자바 키워드는 사용할 수 없음
- '클래스 이름.java'로 소스 파일 생성

Calculator, Car, Member, ChatClient, ChatServer, Web_Browser

```
public class 클래스이름 {  
  
}
```

- 일반적으로 소스파일당 하나의 클래스를 선언함
- 2개 이상의 클래스 선언도 가능하지만 컴파일하면 바이트코드파일은 선언한 클래스 개수만큼 생김.

객체 생성과 클래스 변수

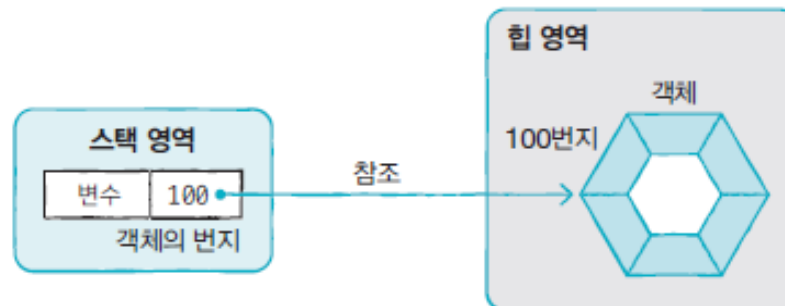
❖ 클래스로부터 객체를 생성

- new 클래스();
- new 연산자로 메모리 힙 영역에 객체 생성
- 객체 생성 후 객체 번지 리턴
 - 클래스 변수에 저장하여 변수 통해 객체 사용 가능



```
클래스 변수;  
변수 = new 클래스();
```

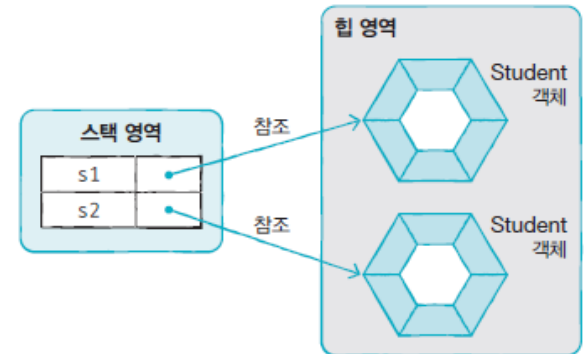
```
클래스 변수 = new 클래스();
```



객체 생성과 클래스 변수

```
public class Student {  
}
```

```
• public class StudentExample {  
    public static void main(String[] args) {  
        Student s1 = new Student();  
        System.out.println("s1 변수가 Student 객체를 참조합니다.")  
  
        Student s2 = new Student();  
        System.out.println("s2 변수가 또 다른 Student 객체를 참조합니다.");  
    }  
}
```



❖ 클래스의 두 용도

- 라이브러리(API : Application Program Interface) 클래스
 - 객체 생성 및 메소드 제공 역할 - Student.java
- 실행 클래스
 - main() 메소드 제공 역할 - StudentExample.java

객체 생성과 클래스 변수-예제

Student.java

```
1 package sec01.exam01;
2
3 public class Student {
4
5 }
6
```

StudentExample.java

```
1 package sec01.exam01;
2
3 public class StudentExample {
4     public static void main(String[] args) {
5         Student s1 = new Student();
6         System.out.println("s1 변수가 Student 객체를 참조합니다.");
7
8         Student s2 = new Student();
9         System.out.println("s2 변수가 또 다른 Student 객체를 참조합니다.");
10    }
11 }
```

클래스의 구성 멤버

❖ 클래스 멤버

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

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

키워드로 끝내는 핵심 포인트

- **클래스**: 객체를 만들기 위한 설계도
- **객체**: 클래스로부터 생성되며 'new 클래스()'로 생성
- **new 연산자**: 객체 생성 연산자이며 생성자 호출하고 객체 생성 번지를 리턴
- **클래스 변수**: 클래스로 선언한 변수이며 해당 클래스의 객체 번지가 저장됨
- **인스턴스**: 객체는 클래스의 인스턴스
- **클래스 멤버**: 클래스에 선언되는 멤버로 필드, 생성자, 메소드가 있음

Chapter

06

클래스

06-2. 필드

❖ 목차

- 시작하기 전에
- 필드 선언
- 필드 사용
- 키워드로 끝내는 핵심 포인트

시작하기 전에

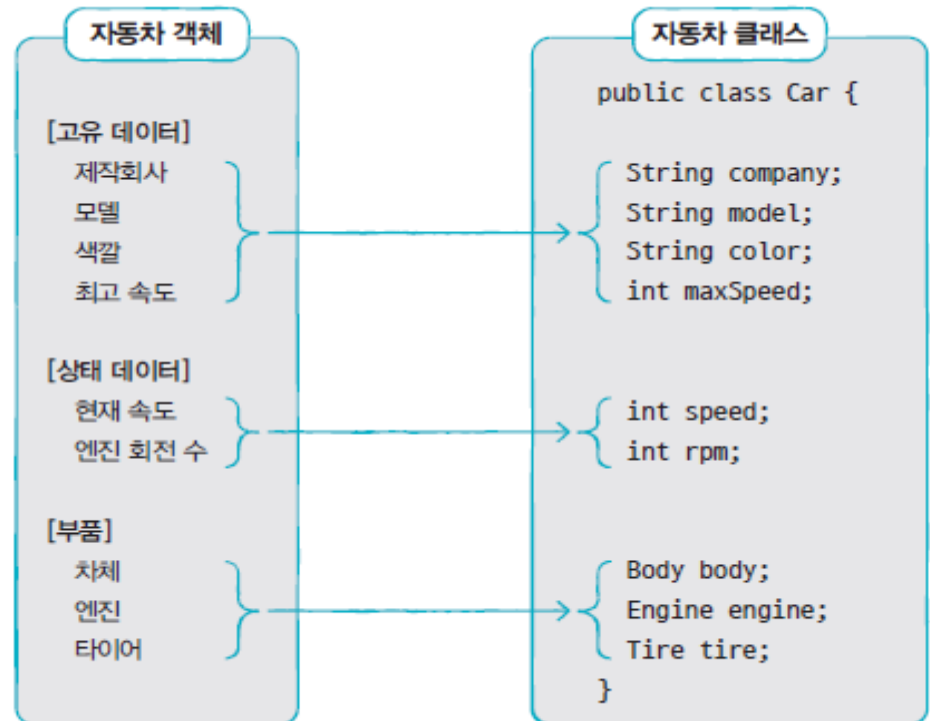
[핵심 키워드] : 필드 선언, 필드 사용

[핵심 포인트]

필드는 객체의 고유 데이터, 부품 객체, 상태 정보를 저장
필드를 선언하고 생성한 뒤 이를 읽고 변경하는 방법을 학습

❖ 필드 (field)

- 객체의 고유 데이터,
객체가 가져야 할 부품,
객체의 현재 상태 데이터 등을 저장



필드 선언

❖ 필드 선언

- 클래스 중괄호 블록 어디서든 존재 가능
- 생성자와 메소드 중괄호 블록 내부에는 선언 불가
- 변수와 선언 형태 유사하나 변수 아님에 주의

- class XXX {

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

선언 형태는 변수와 비슷하지만,
필드를 변수라고 부르지는 않음.

- }

필드 선언

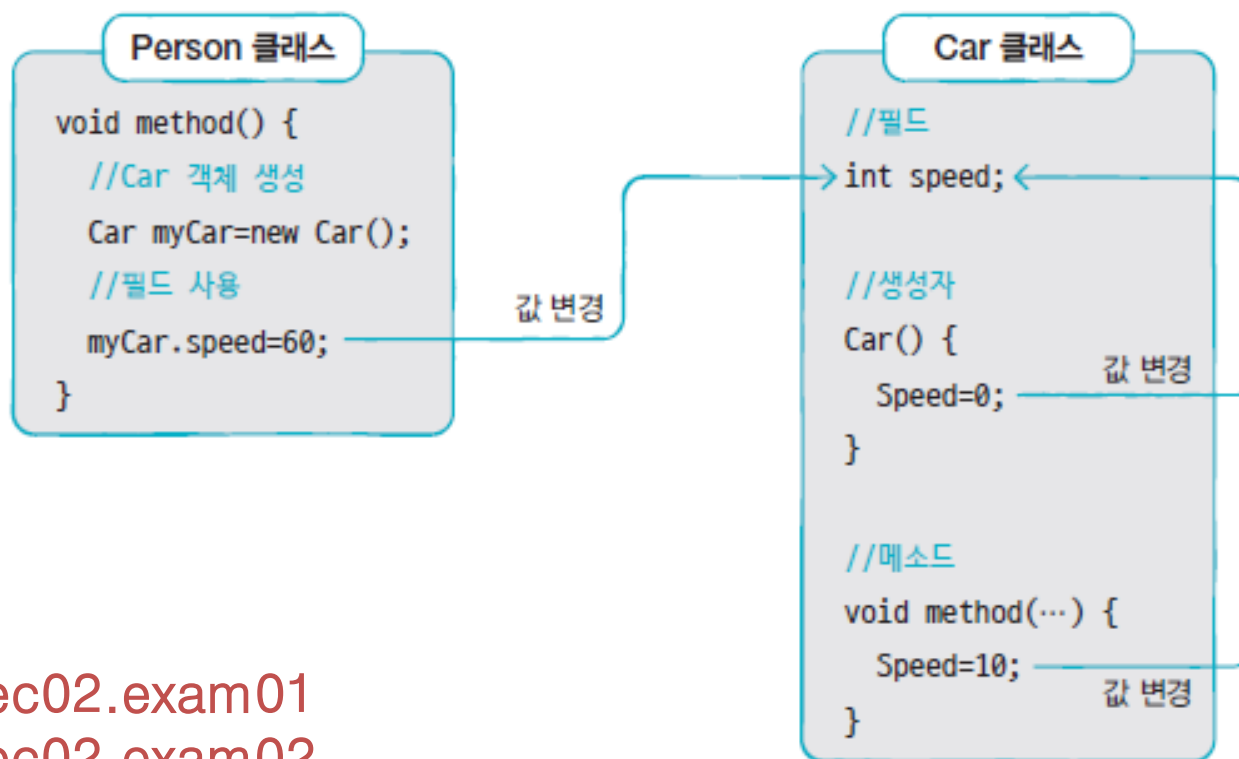
- 초기값은 주어질 수도, 생략할 수도 있음
 - 초기값 지정되지 않은 필드는 객체 생성 시 자동으로 기본 초기값 설정

| 분류 | 타입 | 초기값 |
|-------|-------------------------------|--------------------------------------|
| 기본 타입 | 정수 타입 | byte char short int long |
| | 실수 타입 | float double |
| | 논리 타입 | boolean |
| 참조 타입 | 배열 클래스(String 포함) 인터페이스 | null null null |

필드 사용

❖ 필드 사용

- 필드값 읽고 변경하는 작업
- 클래스 내부 생성자 및 메소드에서 사용하는 경우 : 필드 이름으로 읽고 변경
- 클래스 외부에서 사용하는 경우 : 클래스로부터 객체 생성한 뒤 필드 사용



sec02.exam01
sec02.exam02

필드 사용

myCar 변수가 Car 객체를 참조하도록 객체 생성

(.)연산자를 사용해 company 필드에 접근

Car.java

```
1 package sec02.exam01;
2
3 public class Car {
4     //필드
5     String company = "현대자동차";
6     String model = "그랜저";
7     String color = "검정";
8     int maxSpeed = 350;
9     int speed;
10 }
```

“Car” 클래스 생성 후
“company ~ speed”
까지의 필드 생성

CarExample.java

```
1 package sec02.exam01;
2
3 public class CarExample {
4     public static void main(String[] args) {
5         //객체 생성
6         Car myCar = new Car();
7
8         //필드 값 읽기
9         System.out.println("제작회사: " + myCar.company);
10        System.out.println("모델명: " + myCar.model);
11        System.out.println("색깔: " + myCar.color);
12        System.out.println("최고속도: " + myCar.maxSpeed);
13        System.out.println("현재속도: " + myCar.speed);
14
15        //필드 값 변경
16        myCar.speed = 60;
17        System.out.println("수정된 속도: " + myCar.speed);
18    }
19 }
```

키워드로 끝내는 핵심 포인트

- **필드 선언** : 클래스 중괄호 블록 어디서든 선언하나 생성자나 메소드 내부에서는 사용 불가
- **필드 사용** :
 - 클래스 내부의 생성자와 메소드에서 바로 사용 가능
 - 클래스 외부에서 사용할 경우 반드시 객체 생성하고 참조 변수 통해 사용

Chapter

06

클래스

06-3. 생성자

❖ 목차

- 시작하기 전에
- 기본 생성자
- 생성자 선언
- 필드 초기화
- 생성자 오버로딩
- 다른 생성자 호출 : this()
- 키워드로 끝내는 핵심 포인트

시작하기 전에

[핵심 키워드] : 기본 생성자, 생성자 선언, 매개 변수, 객체 초기화, 오버로딩, this()

[핵심 포인트]

생성자는 new 연산자로 호출되는 중괄호{} 블록이다.
객체 생성 시 초기화를 담당한다.

❖ 생성자 (constructor)

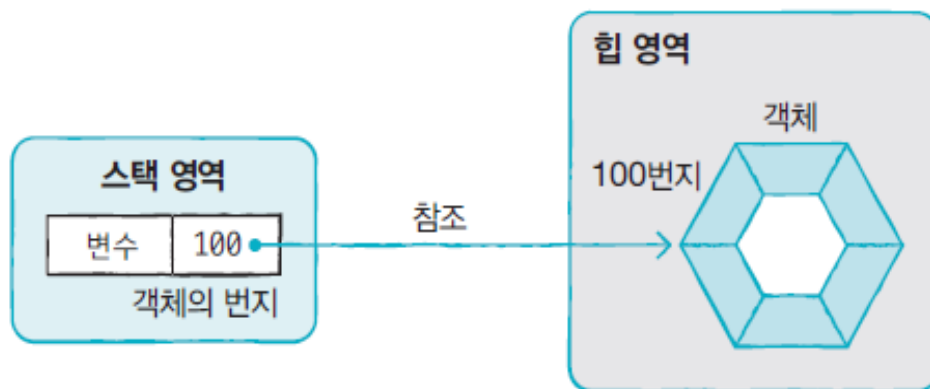
- 클래스로부터 new 연산자로 객체를 생성할 때 호출되어 객체의 초기화를 담당

❖ 객체 초기화

- 필드를 초기화하거나 메소드를 호출해,
객체를 사용할 준비를 하는 것

❖ 생성자가 성공적으로 실행

- 힙 영역에 객체 생성되고 객체 번지가 i



기본 생성자

❖ 기본 생성자 (default constructor)

- 클래스 내부에 생성자 선언 생략할 경우 바이트 코드에 자동 추가

```
[public] 클래스() { }
```

- 클래스에 생성자 선언하지 않아도
new 생성자()로 객체 생성 가능

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

↑
기본 생성자

소스 파일(Car.java)

```
public class Car {  
  
}
```

컴파일

바이트 코드 파일(Car.class)

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

↑
기본 생성자

생성자 선언

❖ 생성자 선언

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

} 생성자 블록

- 매개 변수 선언은 생략할 수도 있고 여러 개 선언할 수도 있음

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

- 클래스에 생성자가 명시적으로 선언되었을 경우 반드시 선언된 생성자 호출하여 객체 생성

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

필드 초기화

❖ 생성자의 필드 초기화

```
public class Korean {  
    //필드  
    String nation = "대한민국";  
    String name;  
    String ssn;  
  
    //생성자  
    public Korean(String n, String s) {  
        name = n;  
        ssn = s;  
    }  
}
```

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

필드 초기화

- 매개 변수 이름 은 필드 이름과 유사하거나 동일한 것 사용 권장
- 필드와 매개 변수 이름 완전히 동일할 경우 this.필드로 표현

```
public Korean(String name, String ssn) {  
    this.name = name;  
    this.ssn = ssn;  
}
```

↑ ↑
필드 매개 변수

↑ ↑
필드 매개 변수

필드 초기화-예제

Korean.java

```
1 package sec03.exam02;
2
3 public class Korean {
4     //필드
5     String nation = "대한민국";
6     String name;
7     String ssn;
8
9     //생성자
10    public Korean(String n, String s) {
11        name = n;
12        ssn = s;
13    }
14
15    /*public Korean(String name, String ssn) {
16        this.name = name;
17        this.ssn = ssn;
18    }*/
19 }
```

Korean생성자의 매개변수 이름은 n, s 사용
이 값들은 각각 name, ssn 필드의 초기값으로 사용됨

KoreanExample.java

```
1 package sec03.exam02;
2
3 public class KoreanExample {
4     public static void main(String[] args) {
5         Korean k1 = new Korean("박자바","011225-1234567");
6         System.out.println("k1.name : " + k1.name);
7         System.out.println("k1.ssn : " + k1.ssn);
8
9         Korean k2 = new Korean("김자바","930525-0654321");
10        System.out.println("k2.name : " + k2.name);
11        System.out.println("k2.ssn : " + k2.ssn);
12    }
13 }
```

생성자 오버로딩

❖ 생성자 오버로딩 (overloading)

- 매개 변수를 달리하는 생성자를 여러 개 선언하는 것을 의미
- 이유 : 외부에서 제공되는 다양한 데이터를 사용하여 객체 화하기 위해

```
public class 클래스 {  
    클래스 ( [타입 매개변수, ...] ) {  
        ...  
    }  
  
    클래스 ( [타입 매개변수, ...] ) {  
        ...  
    }  
}
```

[생성자의 오버로딩]
매개 변수의 타입, 개수, 순서가 다르게 선언

생성자 오버로딩

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

```
Car car1 = new Car();  
Car car2 = new Car("그랜저");  
Car car3 = new Car("그랜저", "흰색");  
Car car4 = new Car("그랜저", "흰색", 300);
```

- 매개 변수의 타입, 개수, 선언된 순서 같은 경우, 매개 변수 이름만 바꾸는 것은 생성자 오버로딩 아님

```
Car(String model, String color) { ... }  
Car(String color, String model) { ... } //오버로딩이 아님
```

다른 생성자 호출: this()

❖ this() 코드

- 생성자에서 다른 생성자 호출시 사용
- 필드 초기화 내용을 한 생성자에만 집중 작성하고 나머지 생성자는 초기화 내용 가진 생성자로 호출
 - 생성자 오버로딩 증가 시 중복 코드 발생 문제 해결

```
클래스( [매개변수, ...] ) {  
    this( 매개변수, ..., 값, ... ); ← 클래스의 다른 생성자 호출  
    실행문;  
}
```

- 생성자 첫 줄에서만 허용

다른 생성자 호출: this()

```
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

```
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;  
}
```

} 공통 실행코드



다른 생성자 호출: this()

- 예제: 다른 생성자 호출해서 중복코드 줄이기

```
1 package sec03.exam04;
2
3 public class Car {
4     //필드
5     String company = "현대자동차";
6     String model;
7     String color;
8     int maxSpeed;
9
10    //생성자
11    Car() {
12    }
13
14    Car(String model) {
15        this(model, null, 0);
16    }
17
18    Car(String model, String color) {
19        this(model, color, 0);
20    }
21
22    Car(String model, String color, int maxSpeed) {
23        this.model = model;
24        this.color = color;
25        this.maxSpeed = maxSpeed;
26    }
27 }
```

```
1 package sec03.exam04;
2
3 public class CarExample {
4     public static void main(String[] args) {
5         Car car1 = new Car();
6         System.out.println("car1.company : " + car1.company);
7         System.out.println();
8
9         Car car2 = new Car("자가용");
10        System.out.println("car2.company : " + car2.company);
11        System.out.println("car2.model : " + car2.model);
12        System.out.println();
13
14        Car car3 = new Car("자가용", "빨강");
15        System.out.println("car3.company : " + car3.company);
16        System.out.println("car3.model : " + car3.model);
17        System.out.println("car3.color : " + car3.color);
18        System.out.println();
19
20        Car car4 = new Car("택시", "검정", 200);
21        System.out.println("car4.company : " + car4.company);
22        System.out.println("car4.model : " + car4.model);
23        System.out.println("car4.color : " + car4.color);
24        System.out.println("car4.maxSpeed : " + car4.maxSpeed);
25    }
26 }
```

키워드로 끝내는 핵심 포인트

- **기본 생성자** : 클래스 선언 시 컴파일러에 의해 자동으로 추가되는 생성자
- **생성자 선언** : 생성자를 명시적으로 선언 가능. 생성자를 선언하면 기본 생성자는 생성되지 않음
- **매개 변수** : 생성자 호출 시 값을 전달받기 위해 선언되는 변수
- **객체 초기화** : 생성자 내부에서 필드값 초기화하거나 메소드 호출해서 사용 준비를 하는 것
- **오버로딩** : 매개 변수 달리하는 생성자를 여러 개 선언
- **this()** : 객체 자신의 또다른 생성자를 호출할 때 사용

Chapter

06

클래스

06-4. 메소드

❖ 목차

- 시작하기 전에
- 메소드 선언
- 리턴 문
- 메소드 호출
- 메소드 오버로딩
- 키워드로 끝내는 핵심 포인트

시작하기 전에

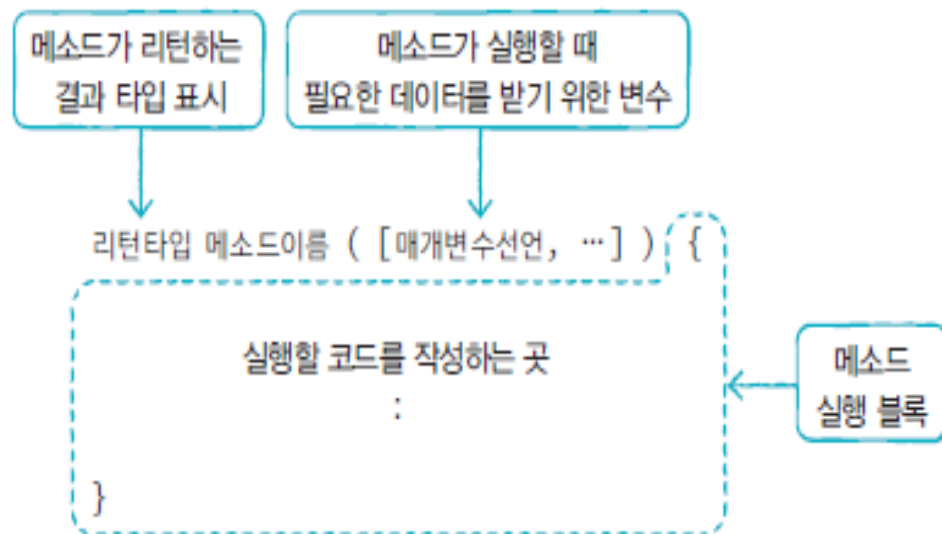
[핵심 키워드] : 선언부, void, 매개 변수, 리턴문, 호출, 오버로딩

[핵심 포인트]

메소드를 선언하고 호출하는 방법에 대해 알아본다.

❖ 메소드 선언부 (signature)

- 리턴 타입 : 메소드가 리턴하는 결과의 타입 표시
- 메소드 이름 : 메소드의 기능 드러나도록 식별자 규칙에 맞게 이름
- 매개 변수 선언 : 메소드 실행할 때 필요한 데이터 받기 위한 변수
- 메소드 실행 블록 : 실행할 코드 작성



메소드 선언

❖ 리턴 타입

- 메소드를 실행한 후의 결과값의 타입
- 리턴값 없을 수도 있음
- 리턴값 있는 경우 리턴 타입이 선언부에 명



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

- 리턴값 존재 여부에 따라 메소드 호출 방법 다름

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

```
int result = divide( 10, 20 ); //컴파일 에러
```

메소드 선언

❖ 메소드 이름

- 숫자로 시작하면 안 되고, \$와 _ 제외한 특수문자 사용 불가
- 메소드 이름은 관례적으로 소문자로 작성
- 서로 다른 단어가 혼합된 이름일 경우 뒤이어 오는 단어의 첫 글자를 대문자로 작성

```
void run() { ... }  
void startEngine() { ... }  
String getName() { ... }  
int[] getScores() { ... }
```

메소드 선언

❖ 매개 변수 선언

- 메소드 실행에 필요한 데이터를 외부에서 받아 저장할 목적

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

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

```
byte b1 = 10;
```

```
byte b2 = 20;
```

```
double result = divide( b1, b2 );
```

- 잘못된 매개값 사용하여 컴파일 에러 발생하는 경우

```
double result = divide( 10.5, 20.0 );
```


메소드 선언

```
1 package sec04.exam01;
2
3 public class Calculator {
4     //메소드
5     void powerOn() {
6         System.out.println("전원을 켭니다.");
7     }
8
9     int plus(int x, int y) {
10         int result = x + y;
11         return result;
12     }
13
14     double divide(int x, int y) {
15         double result = (double)x / (double)y;
16         return result;
17     }
18
19     void powerOff() {
20         System.out.println("전원을 끕니다.");
21     }
22 }
```

```
1 package sec04.exam01;
2
3 public class CalculatorExample {
4     public static void main(String[] args) {
5         Calculator myCalc = new Calculator();
6         myCalc.powerOn();
7
8         int result1 = myCalc.plus(5,6);
9         System.out.println("result1: " + result1);
10
11         byte x = 10;
12         byte y = 4;
13         double result2 = myCalc.divide(x, y);
14         System.out.println("result2: " + result2);
15
16         myCalc.powerOff();
17     }
18 }
```

```
전원을 켭니다.
result1: 11
result2: 2.5
전원을 끕니다
```

메소드 선언

❖ 매개 변수의 개수를 모를 경우

- 매개 변수를 배열 타입으로 선언

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

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

- 배열 생성하지 않고 값의 목록만 넘겨주는 방식

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

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

메소드 선언- 매개 변수의 개수를 모를 경우

```
1 package sec04.exam02;
2
3 public class Computer {
4     int sum1(int[] values) {
5         int sum = 0;
6         for(int i=0; i<values.length; i++) {
7             sum += values[i];
8         }
9         return sum;
10    }
11
12    int sum2(int ... values) {
13        int sum = 0;
14        for(int i=0; i<values.length; i++) {
15            sum += values[i];
16        }
17        return sum;
18    }
19 }
```

```
1 package sec04.exam02;
2
3 public class ComputerExample {
4     public static void main(String[] args) {
5         Computer myCom = new Computer();
6
7         int[] values1={1,2,3};
8         int result1 = myCom.sum1(values1);
9         System.out.println("result1: " + result1);
10
11         int result2 = myCom.sum1(new int[] {1,2,3,4,5});
12         System.out.println("result2: " + result2);
13
14         int result3 = myCom.sum2(1,2,3);
15         System.out.println("result3: " + result3);
16
17         int result4 = myCom.sum2(1,2,3,4,5);
18         System.out.println("result4: " + result4);
19     }
20 }
```

리턴(return)문

❖ 리턴값이 있는 메소드

- 메소드 선언에 리턴 타입 있는 메소드는 리턴문 사용하여 리턴값 지정

```
return 리턴값;
```

- return문의 리턴값은 리턴타입이거나 리턴타입으로 변환될 수 있어야 함

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

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


리턴(return)문

❖ 리턴값이 없는 메소드 : void

- void 선언된 메소드에서 return문 사용하여 메소드 실행 강제

```
return;
```

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



리턴(return)문

```
1 package sec04.exam03;
2
3 public class Car {
4     //필드
5     int gas;
6
7     //생성자
8
9     //메소드
10 void setGas(int gas) {
11     this.gas = gas;
12 }
13
14 boolean isLeftGas() {
15     if(gas==0){
16         System.out.println("gas가 없습니다.");
17         return false;
18     }
19     System.out.println("gas가 있습니다.");
20     return true;
21 }
22
23
24 void run() {
25     while(true) {
26         if(gas > 0){
27             System.out.println("달립니다.(gas잔량:" + gas + ")");
28             gas -= 1;
29         } else {
30             System.out.println("멈춥니다.(gas잔량:" + gas + ")");
31             return;
32         }
33     }
34 }
35 }
```

리턴값이 없는 메소드로 매개값을 받아서 gas 필드값을 생성

리턴값이 boolean인 메소드로 gas 필드값이 0이면 false를, 0이 아니면 true를 리턴

리턴값이 없는 메소드로 gas 필드값이 0이면 return문으로 메소드를 강제 종료함(값을 return 하는 것이 아님)

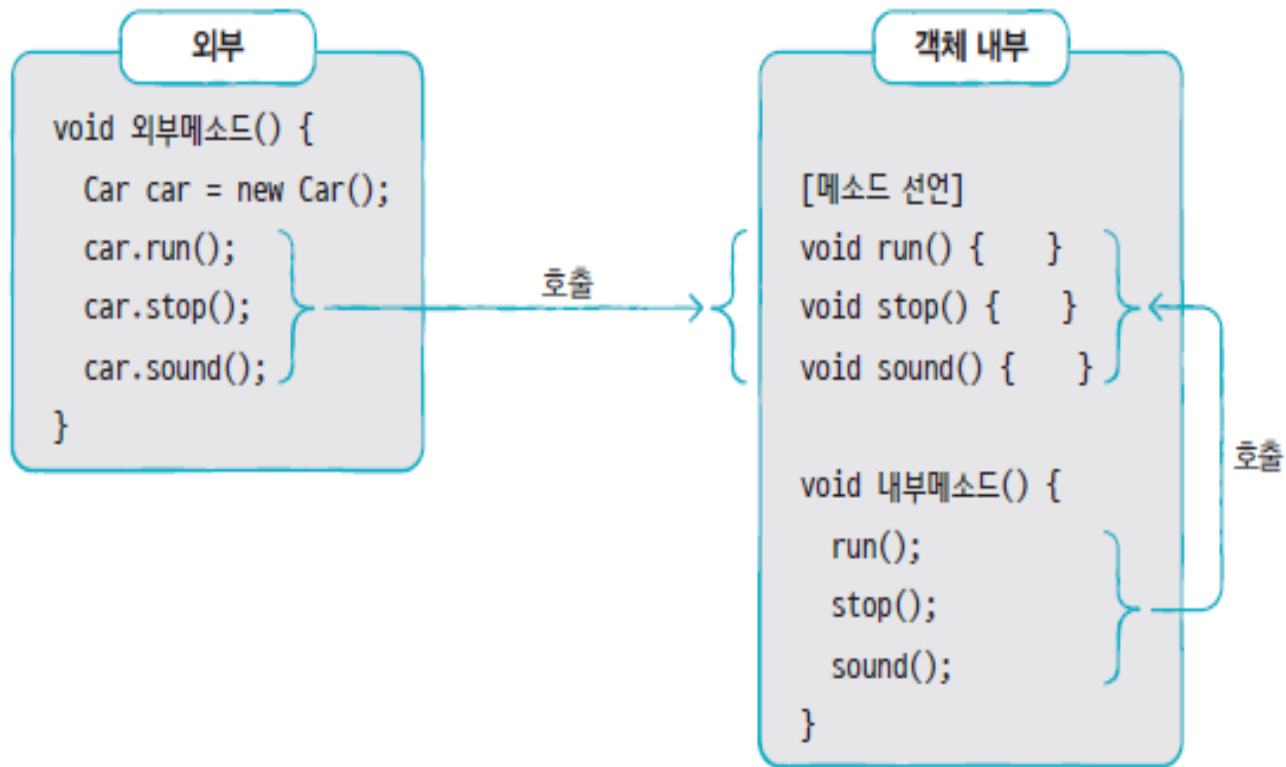
리턴(return)문

```
1 package sec04.exam03;
2
3 public class CarExample {
4     public static void main(String[] args) {
5         Car myCar = new Car();
6
7         myCar.setGas(5); //Car의 setGas() 메소드 호출
8
9         boolean gasState = myCar.isLeftGas(); //Car의 isLeftGas() 메소드 호출
10        if(gasState) {
11            System.out.println("출발합니다.");
12            myCar.run(); //Car의 run() 메소드 호출
13        }
14
15        if(myCar.isLeftGas()) { //Car의 isLeftGas() 메소드 호출
16            System.out.println("gas를 주입할 필요가 없습니다.");
17        } else {
18            System.out.println("gas를 주입하세요.");
19        }
20    }
21 }
```

메소드 호출

❖ 메소드 호출

- 클래스 내외부의 호출에 의해 메소드 실행
 - 내부의 경우 단순히 메소드 이름으로 호출
 - 외부의 경우 클래스로부터 객체 생성한 뒤 참조 변수 사용하여 메소드 호출



메소드 호출

❖ 객체 내부에서 호출

- 메소드가 리턴값 없거나(void) 있어도 받고 싶지 않은 경우

메소드(매개값, ...);

```
public class ClassName {  
    void method1( String p1, int p2 ) {  
        ↓ ② 실행  
    }  
  
    void method2() {  
        method1( "홍길동", 100 );  
    }  
}
```

① 호출

메소드 호출

- 리턴값 있는 메소드 호출하고 리턴값 받고 싶은 경우

타입 변수 = 메소드(매개값, ...);



```
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  
    }  
}
```

메소드 호출 예제 - 클래스 내부에서 메소드 호출

```
1 package sec04.exam04;
2
3 public class Calculator {
4     int plus(int x, int y) {
5         int result = x + y;
6         return result;
7     }
8
9     double avg(int x, int y) {
10         double sum = plus(x, y);
11         double result = sum / 2;
12         return result;
13     }
14
15     void execute() {
16         double result = avg(7,10);
17         println("실행결과: " + result);
18     }
19
20     void println(String message) {
21         System.out.println(message);
22     }
23 }
```

```
1 package sec04.exam04;
2
3 public class CalculatorExample {
4     public static void main(String[] args) {
5         Calculator myCalc = new Calculator();
6         myCalc.execute();
7     }
8 }
```

Calculator 객체의 execute() 메소드를 호출하면 아래의 과정으로 실행됨

1. 16라인에서 avg() 메소드 실행
2. avg()메소드의 10라인에서 호출이 일어나 plus() 메소드 실행됨
3. plus() 메소드가 리턴값을 주면 avg() 메소드는 10라인에서 리턴값을 sum 변수에 저장하고 11라인을 실행한 후 12라인에서 execute() 메소드로 리턴값을 줌
4. execute() 메소드는 16라인에서 avg() 메소드의 리턴값을 받아 17라인에서 println() 메소드를 호출할 때 매개값으로 넘겨줌
5. println() 메소드는 매개값으로 받은 문자열을 21라인에서 콘솔로 출력
6. 마지막으로 execute() 메소드는 18라인을 만나게 되고 종료됨

메소드 호출

❖ 객체 외부에서 호출

- 우선 클래스로부터 객체 생성

```
클래스 참조변수 = new 클래스( 매개값, ... );
```

- 참조 변수와 도트 연산자 사용하여 메소드 호출

```
참조변수.메소드( 매개값, ... ); //리턴값이 없거나, 있어도 리턴값을 받지 않을 경우  
타입 변수 = 참조변수.메소드( 매개값, ... ); //리턴값이 있고, 리턴값을 받고 싶을 경우
```

```
Car myCar = new Car();  
myCar.keyTurnOn();  
myCar.run();  
int speed = myCar.getSpeed();
```

메소드 호출 예제 - 클래스 외부에서 메소드 호출

```
1 package sec04.exam05;
2
3 public class Car {
4     //필드
5     int speed;
6
7     //생성자
8
9     //메소드
10    int getSpeed() {
11        return speed;
12    }
13
14    void keyTurnOn() {
15        System.out.println("키를 돌립니다.");
16    }
17
18    void run() {
19        for(int i=10; i<=50; i+=10){
20            speed = i;
21            System.out.println("달립니다.(시속:" + speed + "km/h)");
22        }
23    }
24 }
```

```
1 package sec04.exam05;
2
3 public class CarExample {
4     public static void main(String[] args) {
5         Car myCar = new Car();
6         myCar.keyTurnOn();
7         myCar.run();
8         int speed = myCar.getSpeed();
9         System.out.println("현재 속도: " + speed + "km/h");
10    }
11 }
```

메소드 오버로딩

❖ 메소드 오버로딩 (overloading)

- 같은 이름의 메소드를 여러 개 선언
- 매개값을 다양하게 받아 처리할 수 있도록 하기 위함
- 매개 변수의 타입, 개수, 순서 중 하나가 달라야

```
class 클래스 {  
    리턴 타입 메소드이름 ( 타입 변수, ... ) { ... }  
    ↑           ↑           ↑  
    동일       동일       매개 변수의 타입, 개수, 순서가 달라야 함  
    ↓           ↓           ↓  
    리턴 타입 메소드이름 ( 타입 변수, ... ) { ... }  
}
```

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

메소드 오버로딩

- 오버로딩된 메소드 호출하는 경우 JVM은 매개값 타입 보고 메소드를 선택

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

- plus(int x, int y)가 실행

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

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

- plus(double x, double y)가 실행

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

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

- plus(double x, double y)가 실행

메소드 오버로딩

- 매개 변수의 타입, 개수, 순서 같은 경우 매개변수 이름 달라도 메소드 오버로딩 아님에 주의

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

- `System.out.println()` 메소드

```
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 예제

```
1 package sec04.exam06;
2
3 public class Calculator {
4     //정사각형의 넓이
5     double areaRectangle(double width) {
6         return width * width;
7     }
8
9     //직사각형의 넓이
10    double areaRectangle(double width, double height) {
11        return width * height;
12    }
13 }
```

```
1 package sec04.exam06;
2
3 public class CalculatorExample {
4     public static void main(String[] args) {
5         Calculator myCalcu = new Calculator();
6
7         //정사각형의 넓이 구하기
8         double result1 = myCalcu.areaRectangle(10);
9
10        //직사각형의 넓이 구하기
11        double result2 = myCalcu.areaRectangle(10,20);
12
13        //결과 출력
14        System.out.println("정사각형 넓이=" + result1);
15        System.out.println("직사각형 넓이=" + result2);
16    }
17 }
```

키워드로 끝내는 핵심 포인트

- **선언부** : 리턴 타입, 메소드 이름, 매개 변수 선언
- **void** : 리턴값이 없는 메소드는 리턴 타입으로 void를 기술해야 함
- **매개 변수** : 메소드 호출 시 제공되는 매개값이 대입되어 메소드 블록 실행 시 이용됨
- **리턴문** : 메소드의 리턴값을 지정하거나 메소드 실행 종료를 위해 사용할 수 있음.
- **호출** : 메소드를 실행하려면 '메소드 이름(매개값. ...)' 형태로 호출
- **오버로딩** : 클래스 내에 같은 이름의 메소드 여러 개 선언하는 것을 말함

Thank You !