클래스와 객체

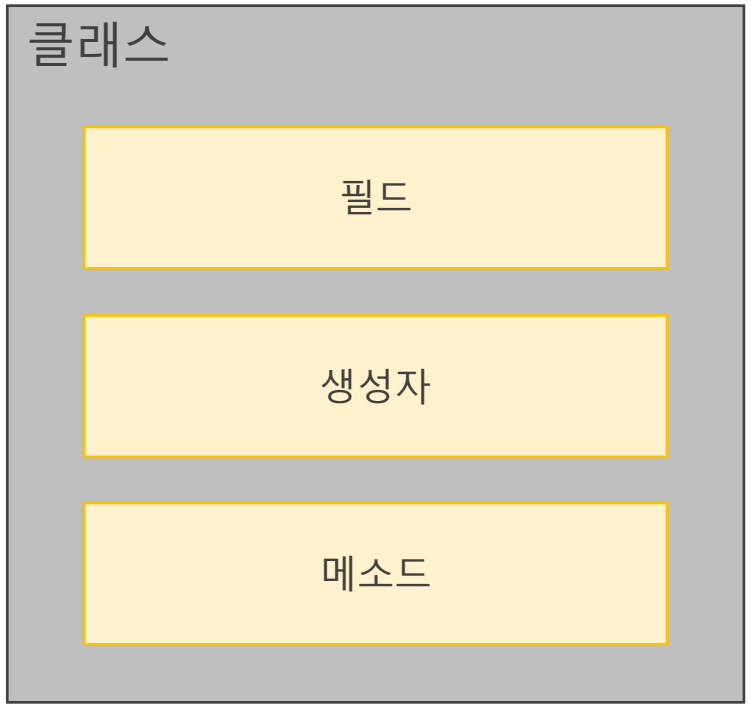
클래스(Class)

- 클래스(Class)
  - ✓ 객체(Object)를 만들어 내기 위한 설계도
  - ✓ 객체의 속성(Attribute)과 행동(Behavior)을 포함하고 있음
  - ✓ 클래스는 객체 내부의 값을 저장하기 위한 필드(Field)와 객체의 기능을 나타내기 위한 메소드(Method)로 구성됨

- 필드(Field)
  - ✓ 객체의 속성을 나타내는 변수

- 생성자(Constructor)
  - ✓ 객체를 생성하는 특별한 메소드

- 메소드(Method)
  - ✓ 객체의 행동을 나타내는 함수
  - ✓ 객체의 기능을 의미함



객체(Object)

- 객체(Object)
  - ✓ 클래스를 이용해서 생성한 실체가 있는 존재
  - ✓ 컴퓨터 메모리 공간을 차지하고 있음
  - ✓ 객체를 인스턴스(Instance)라고도 함
  - ✓ 다른 객체들과 정보를 주고 받는 상호 작용이 가능
  - ✓ 현실 세계의 모든 것이 객체임

모두 객체

컴퓨터

휴대폰

카메라

자동차

사람

클래스와 객체

- **클래스(Class) 예시**

1. 탱크
2. 사람
3. 붕어빵 프라이팬

- **객체(Object) 예시**

1. 탱크 공장에서 만든 새 탱크 10대
2. 교실에 있는 학생 20명
3. 방금 만든 붕어빵 10개

하나의 클래스를 정의하면

여러 객체를 만들 수 있다!

클래스

객체

객체 생성 방법

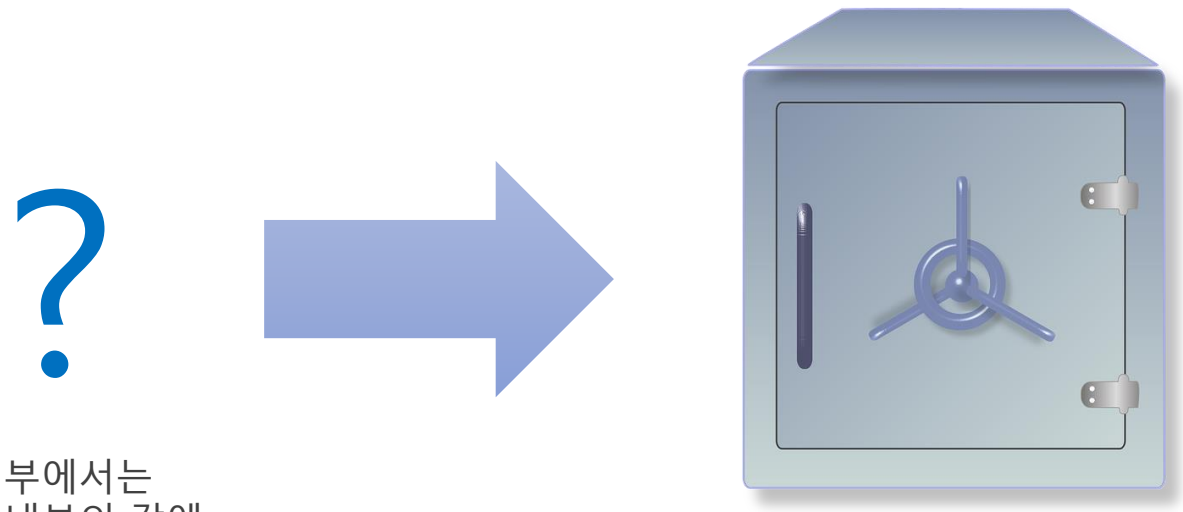
- 일반적으로 new 키워드를 이용해서 생성
- 객체는 메모리를 할당 받아 물리적으로 존재하게 되는 인스턴스화(Instantiate) 과정을 거치는데 이런 이유로 객체를 인스턴스(Instance)라고도 함
- 객체는 클래스를 데이터타입으로 가진다고 생각하면 됨
- 객체를 생성한 뒤 해당 객체를 이용해서 클래스에 만들어 둔 필드나 메소드를 사용할 수 있음

객체 생성 예시

```
Date now = new Date();
```

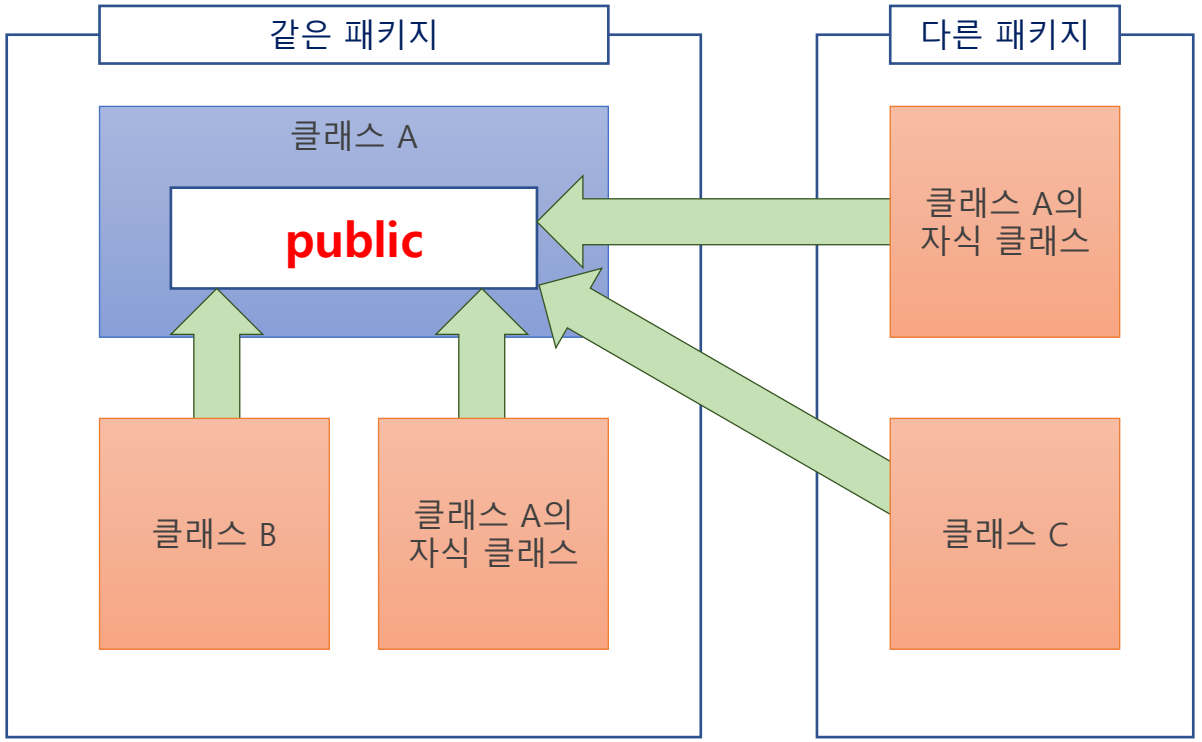
정보 은닉(Information Hiding)

- 정보 은닉(Information Hiding)
  - ✓ 객체 내부의 정보를 다른 객체에게 보여주지 않고 숨기는 것을 의미함
  - ✓ 객체 외부에서는 객체 내부 정보를 직접 접근하거나 조작하는 것이 불가능함
  - ✓ 객체의 정보는 필드에 저장하기 때문에 정보 은닉의 실제 의미는 필드 값을 숨기는 것을 의미함
  - ✓ 필드에 직접 접근하는 대신 객체 외부에서 필드에 접근할 수 있도록 별도의 메소드를 구현함
  - ✓ 자바는 4가지 접근 제어자(private, default, protected, public)를 통해서 정보 은닉을 구현



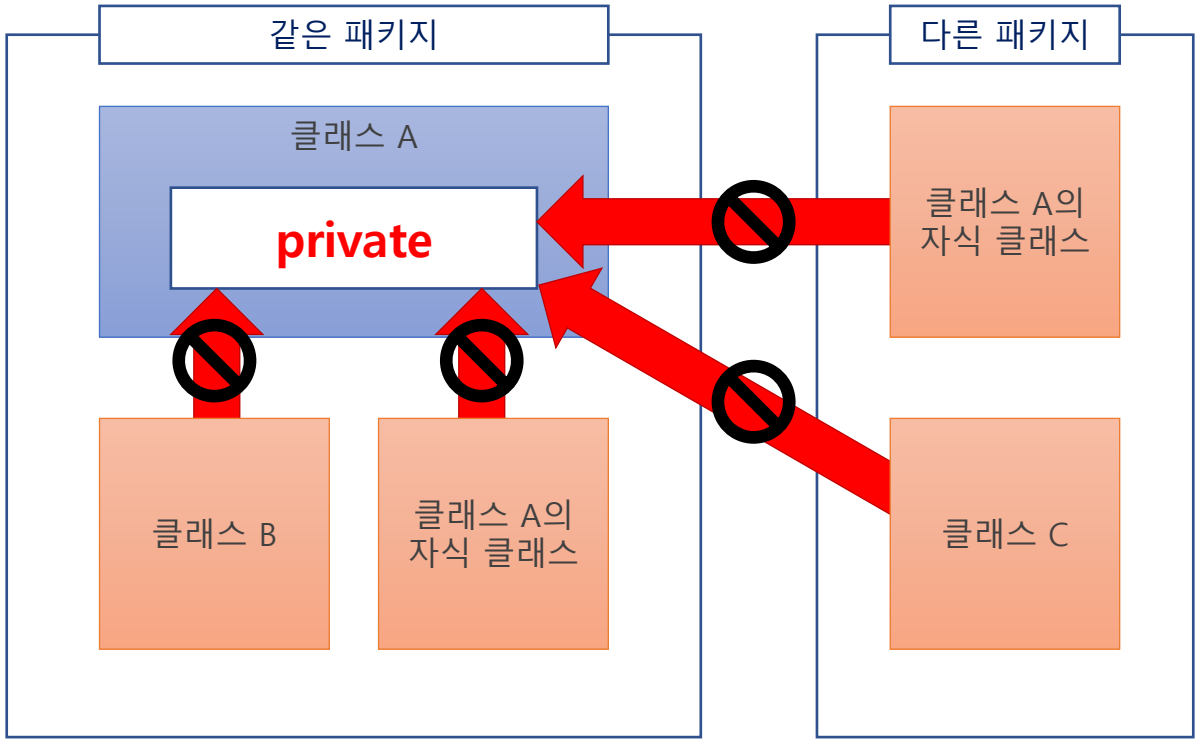
접근 제어자(Access Modifier)

- public
  - ✓ 누구나 접근할 수 있는 권한
  - ✓ 일반적으로 메소드가 public 권한을 가짐



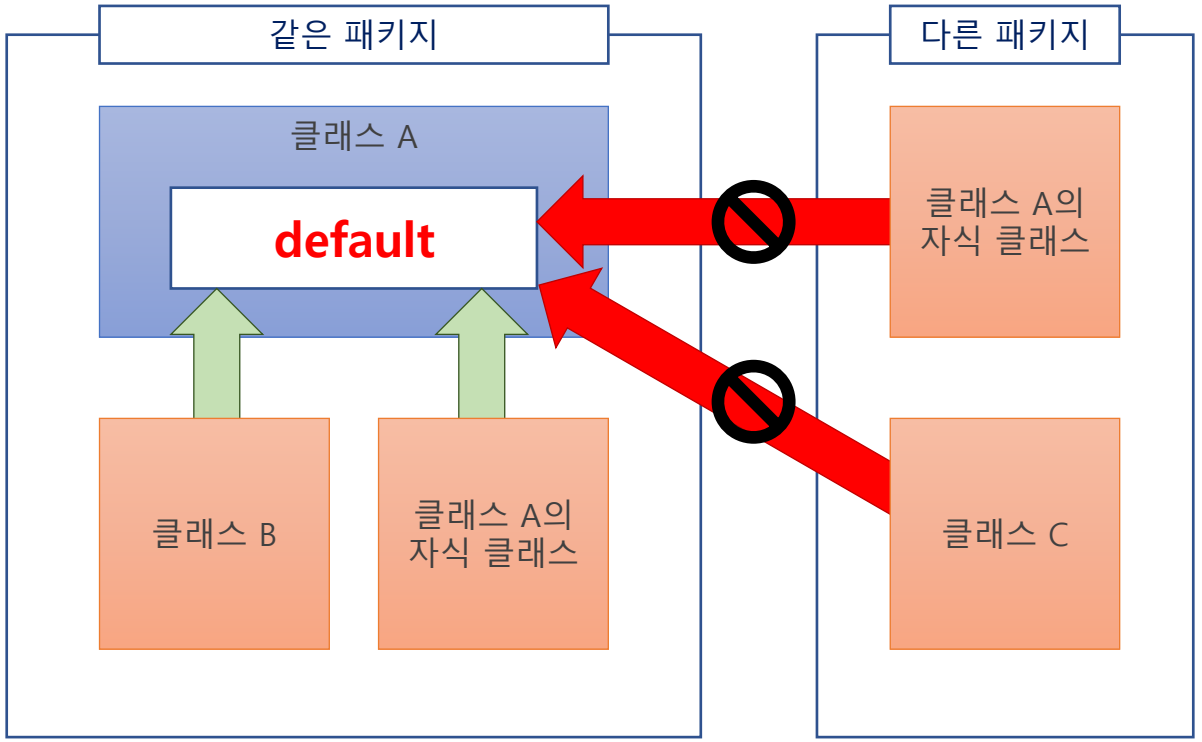
접근 제어자(Access Modifier)

- private
  - ✓ 클래스 내부에서만 접근이 가능한 권한
  - ✓ 일반적으로 필드가 public 권한을 가짐



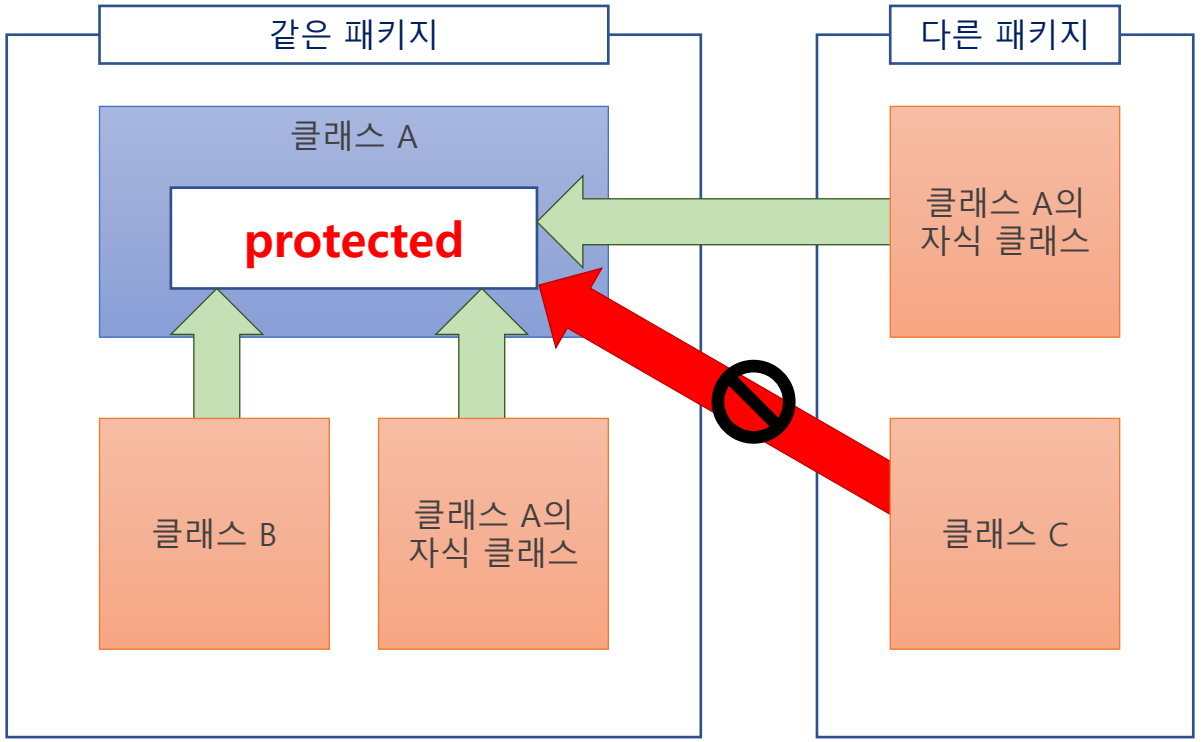
접근 제어자(Access Modifier)

- default
  - ✓ 같은 패키지에서만 접근할 수 있는 권한
  - ✓ 접근 제어자를 생략하면 default로 동작함



접근 제어자(Access Modifier)

- protected
  - ✓ 같은 패키지 또는 자식 클래스에서만 접근할 수 있는 권한
  - ✓ 다른 패키지에 있더라도 자식 클래스는 접근이 가능



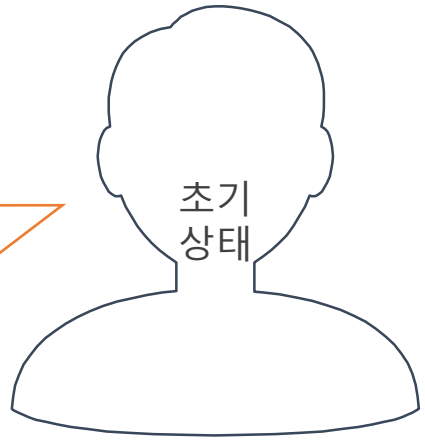
필드(Field)

- 객체(Object)의 속성(Attribute)을 저장하는 변수를 의미
- 일반 변수와 달리 자동으로 초기화 됨
  - ✓ boolean 타입 필드 : false로 초기화
  - ✓ 숫자 타입(int, double 등) 필드 : 0으로 초기화
  - ✓ 참조 타입(String 등) 필드 : null로 초기화

객체마다

필드(속성)값이 다름

모든 객체는 공통적으로
이름/나이/직책/결혼유무
필드(속성)를 가짐



필드 초기화

이름 : null
나이 : 0
직책 : null
결혼유무 : false

객체1

이름 : 전지현
나이 : 35
직책 : 팀장
결혼유무 : true

객체2

이름 : 소지섭
나이 : 30
직책 : 과장
결혼유무 : false

객체3

이름 : 황정민
나이 : 40
직책 : 이사
결혼유무 : true

객체4

이름 : 배수지
나이 : 27
직책 : 주임
결혼유무 : false

필드(Field)

- Person 클래스의 예시
  - ✓ 이름, 나이, 직책, 결혼유무를 저장할 수 있는 필드를 Person 클래스에 선언
  - ✓ Person 클래스 내부에서만 접근할 수 있도록 private 접근 지시자를 부여

```
public class Person {

    private String name;
    private int age;
    private String position;
    private boolean isMarried;

}
```

메소드(Method)

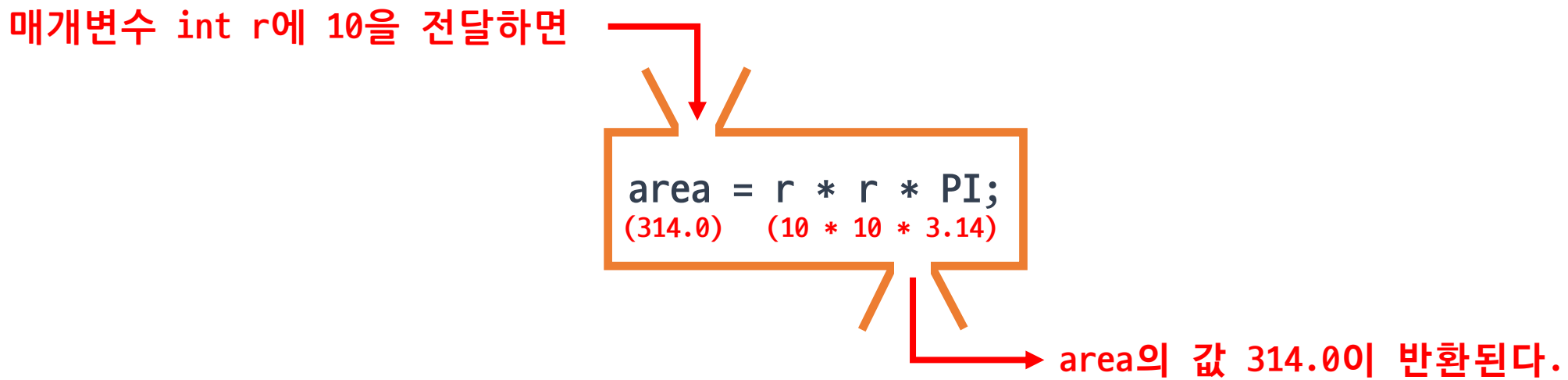
- 객체(Object)의 행동(Behavior)을 의미하는 함수를 의미
- 함수(Function)
  - ✓ 독립적인 기능을 수행하는 프로그램의 단위
  - ✓ 기능별로 구분한 프로그램 단위로 재사용이 가능함
- 형식

```
접근권한 반환타입 메소드명(매개변수) {
    실행코드
    return 반환값
}
```

메소드(Method)

- 예시) 반지름을 전달하면 원의 넓이를 반환하는 getCircleArea 메소드

```
public double getCircleArea(int r) {
    final double PI = 3.14;
    double area = PI * r * r;
    return area;
}
```



메소드의 4가지 유형

- 반환값 X, 매개변수 X

```
void method() {
    ...
}
```

- 반환값 O, 매개변수 X

```
int method() {
    ...
    return 반환값;
}
```

- 반환값 X, 매개변수 O

```
void method(int a) {
    ...
}
```

- 반환값 O, 매개변수 O

```
int method(int a) {
    ...
    return 반환값;
}
```

this

- this
  - ✓ 객체 자신의 참조값(Reference)
  - ✓ 클래스 내부에서만 사용 가능
- this.필드 또는 this() 형태로 사용
- this.필드
  - ✓ 매개변수와 필드명이 동일한 경우 이를 구분하기 위해서 사용
  - ✓ this의 가장 주된 사용처
- this()
  - 생성자 내부에서 사용
  - 같은 클래스 내의 다른 생성자를 호출할 때 사용

```
public class Person {

    private String name;

    public void setName(String name) {
        this.name = name;
    }

}
```

Getter / Setter

- 필드는 private하기 때문에 외부에서 직접 접근하는 것이 불가능
- 외부에서 필드에 접근할 수 있도록 만들어 두는 메소드
- Getter
  - ✓ 필드값을 외부로 반환하는 메소드
  - ✓ "get + 필드명"의 메소드명을 가짐(boolean 타입의 필드는 is + 필드명)
- Setter
  - ✓ 외부로부터 전달 받은 값을 필드에 저장하는 메소드
  - ✓ "set + 필드명"의 메소드명을 가짐
  - ✓ 매개변수명과 필드명을 동일하게 구성하기 때문에 this가 사용됨
- Getter/Setter의 이름은 반드시 정해진 이름을 사용해야 함

```
public class Person {

    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

}
```

메소드 오버로딩(Method Overloading)

- 메소드 오버로딩 : 이름이 같은 메소드를 여러 개 만들 수 있다.
- 오버로딩 조건
  - ✓ 메소드 이름이 동일해야 한다.
  - ✓ 매개변수가 달라야 한다.
  - ✓ 반환타입은 상관이 없다.

성공

```
public int add(int a, int b) {
    return a + b;
}
public int add(int a, int b, int c) {
    return a + b + c;
}
```

메소드 이름이 같고, 매개변수가 다르다.

실패

```
public int add(int a, int b) {
    return a + b;
}
public double add(int a, int b) {
    return (double)(a + b);
}
```

반환타입이 다르지만, 메소드 이름과 매개변수가 모두 같다.

Java

생성자(Constructor)

- 객체(Object)를 생성할 때 사용되는 특별한 메소드
- 생성자 특징
  - ✓ 클래스의 이름과 메소드의 이름이 같음
  - ✓ 반환타입이 없음
  - ✓ 객체가 생성될 때(new) 자동으로 한 번만 호출됨
  - ✓ 매개변수 처리는 일반 메소드와 동일함
  - ✓ 매개변수로 전달된 값을 이용해서 객체의 필드값을 초기화 할 때 사용함
- 형식

```
접근권한 생성자명(매개변수) {
    실행코드
}
```

```java
public class Person {

    // Person 생성자
    public Person() {

    }

}
```

생성자(Constructor)

- 생성자는 객체 생성 방법을 정의하는 메소드
- 생성자의 개수만큼 객체 생성 방법이 존재함
  (생성자가 2개면 객체 생성 방법도 2가지)

```
public class Person {

    private String name;

    public Person() {

    }

    public Person(String n) {
        name = n;
    }

}
```

첫 번째 객체 생성

```
Person p1 = new Person();
```

두 번째 객체 생성

```
Person p2 = new Person("송혜교");
```

디폴트 생성자(Default Constructor)

- 클래스에서 생성자를 만들지 않으면 자동으로 디폴트 생성자가 사용됨
- 객체 생성 역할만 가지고 있으며 필드 초기화는 불가능함
- 생성자를 하나라도 만든다면 디폴트 생성자는 사용할 수 없음

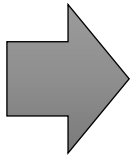
```
public class Person {

    private String name;

    public void walk() {
        System.out.println("Walking");
    }

}
```

생성자가 없는 Person 클래스



```
public class Person {

    private String name;

    public Person() {

    }

    public void walk() {
        System.out.println("Walking");
    }

}
```

디폴트 생성자가
자동으로 사용됨

static vs non-static

➢ static 멤버

- 클래스당 하나만 존재
- 클래스가 로드될 때 메모리 할당
- 모든 객체가 공유
- static 멤버 내부에서는 static 멤버만 사용 가능
- 클래스 멤버라고 함
- 클래스를 이용해서 접근
  (객체를 이용한 접근도 허용되나 비추천)
  - ✓ Math.random()
  - ✓ System.out.println()
  - ✓ Calendar.getInstance()

➢ non-static 멤버

- 객체(인스턴스)마다 존재
- 객체가 생성될 때 메모리 할당
- 각 객체가 개별 값을 가짐
- non-static 멤버는 static 멤버도 사용 가능
- 인스턴스 멤버라고 함
- 객체를 이용해서 접근
  - ✓ Person p = new Person();
  - ✓ p.setName("송혜교");