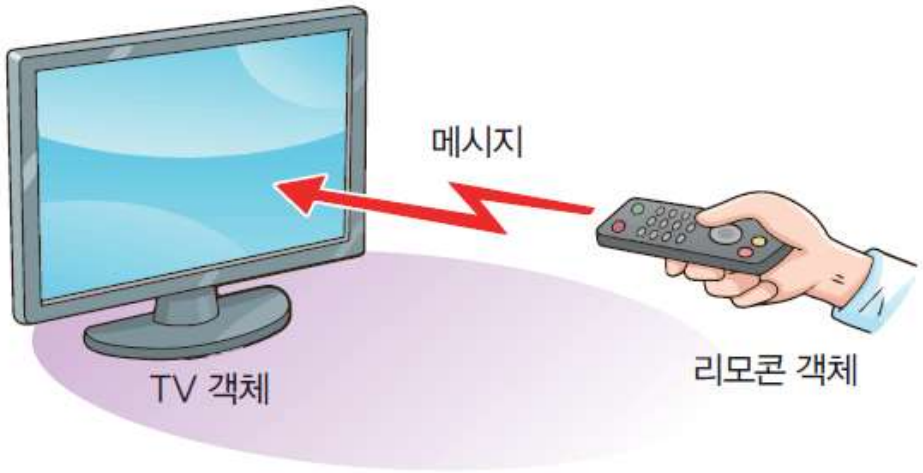
어서와 JAVA는 처음이지!

제5장 클래스, 객체, 메소드

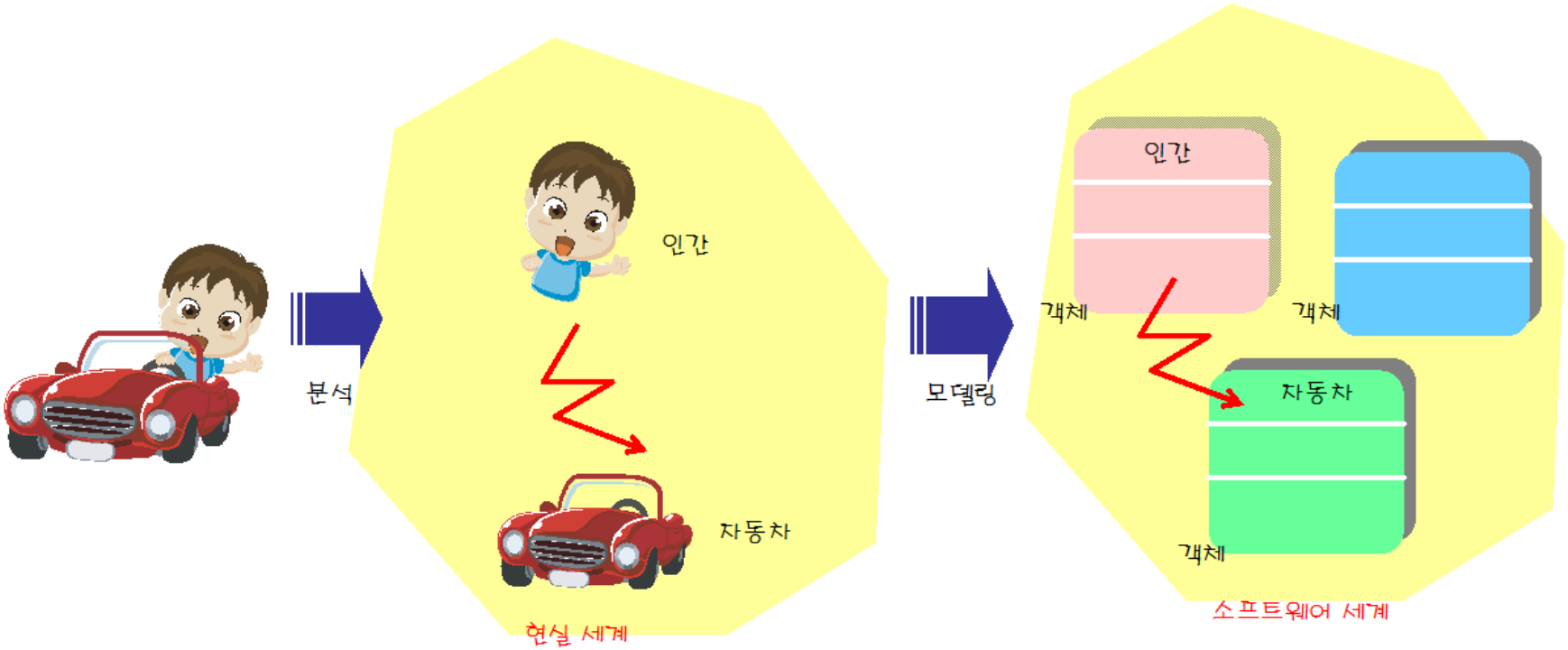

실제 세계는 객체로 이루어진다.
객체
객체
객체
객체
실제 세계는 객체들로
이루어져 있죠!

객체와 메시지



객체 지향이란?

- 실제 세계를 모델링하여 소프트웨어를 개발하는 방법



객체

TV 객체

상태

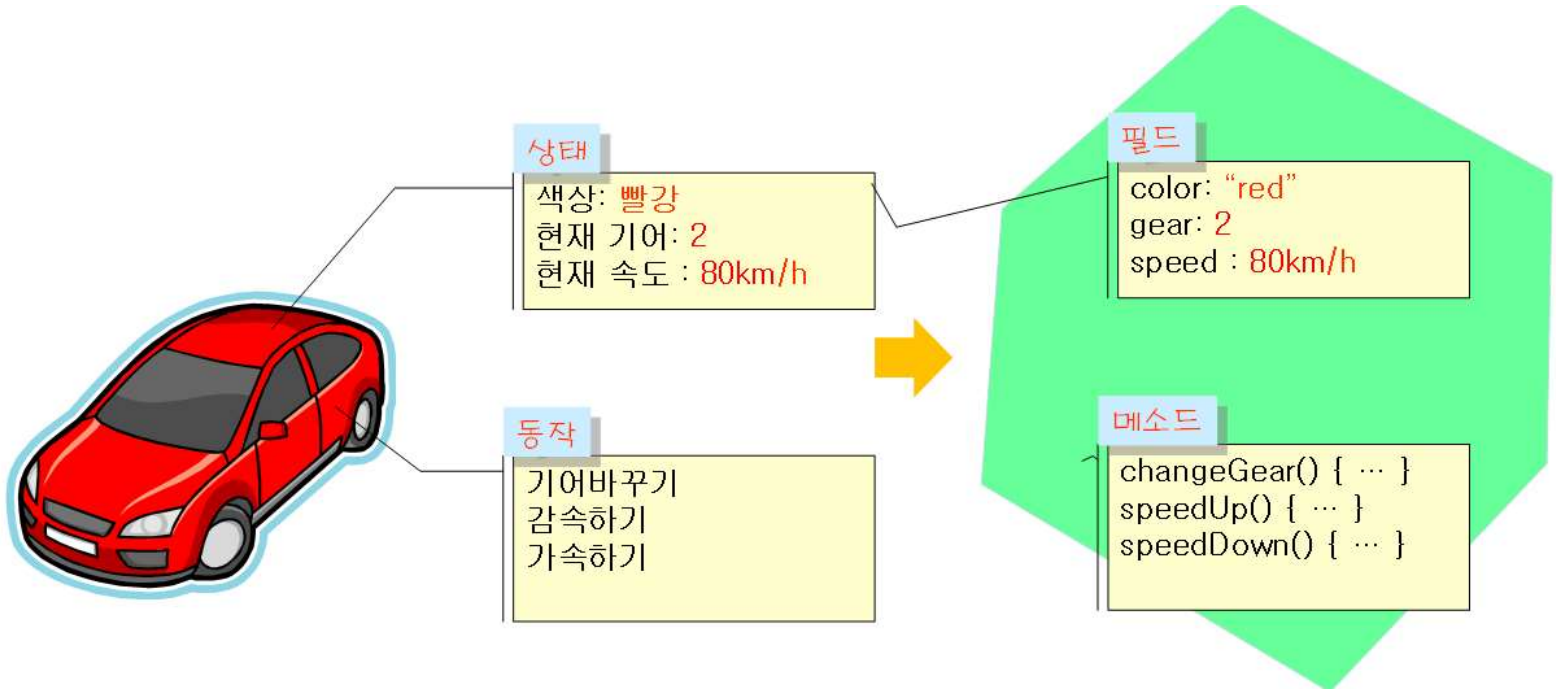
- 채널번호
- 볼륨
- 전원상태

동작

- 켜기
- 끄기
- 채널변경하기
- 볼륨변경하기

객체란?

- **객체(Object)**는 상태와 동작을 가지고 있다.
- 객체의 **상태(state)**는 객체의 특징값(속성)이다.
- 객체의 **동작(behavior)** 또는 행동은 객체가 취할 수 있는 동작



필드와 메소드

TV 객체

필드

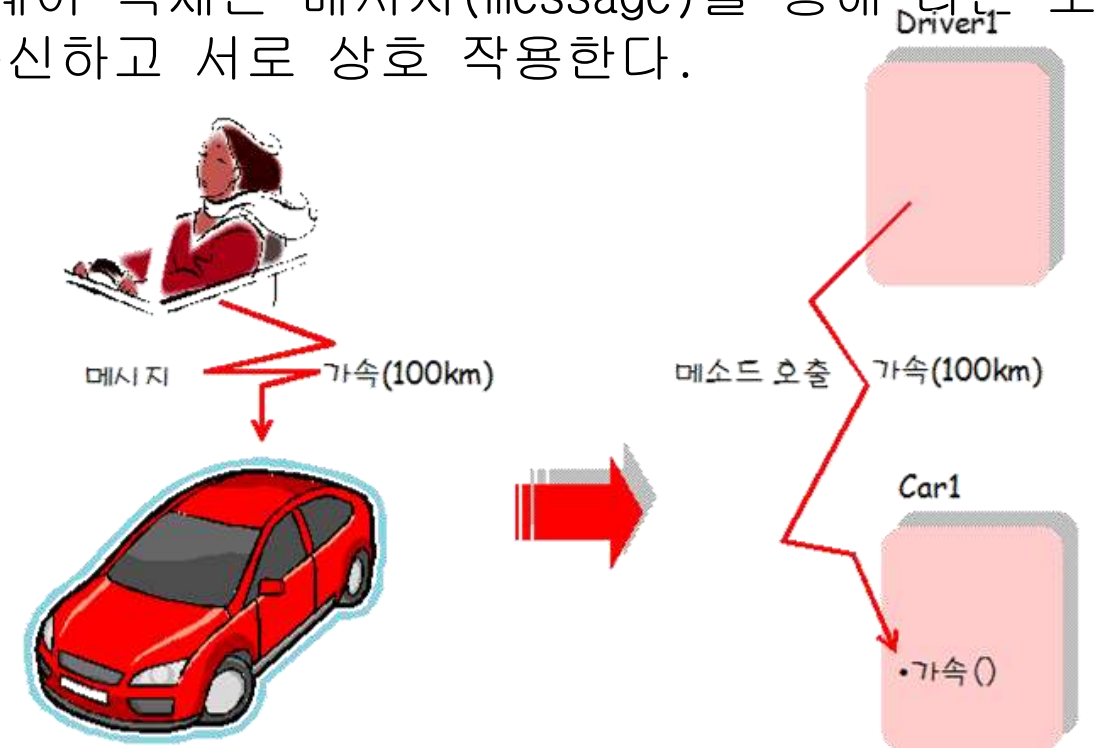
- int channelNO;
- int volume;
- bool on Off

메소드

- turnOn()
- turnOff()
- changeChannel()
- change Volume()

메시지

- 소프트웨어 객체는 메시지(message)를 통해 다른 소프트웨어 객체와 통신하고 서로 상호 작용한다.



객체 지향의 3대 특징

- 캡슐화
- 상속
- 다형성

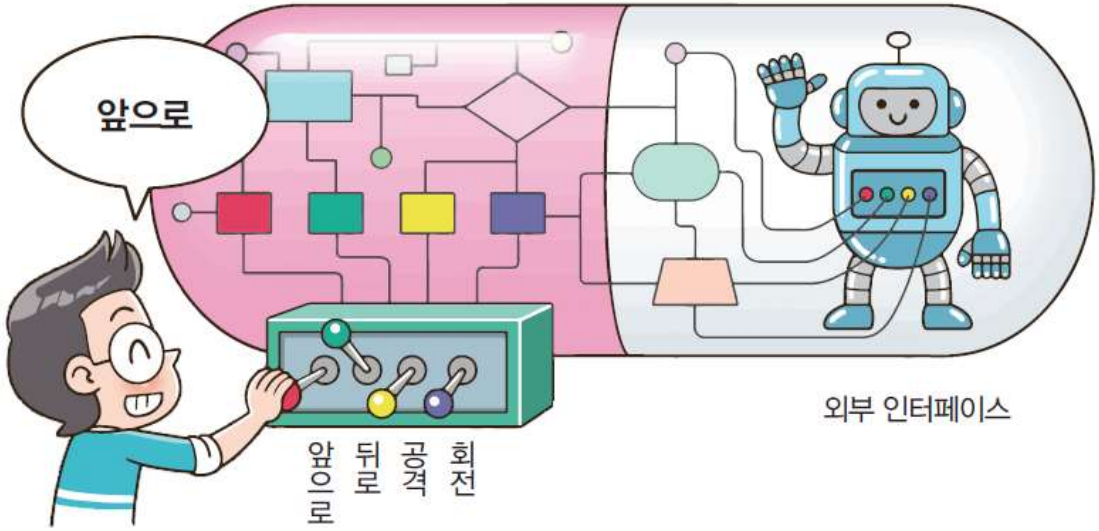
캡슐화

- **캡슐화(encapsulation)**: 관련된 데이터와 알고리즘(코드)이 하나의 묶음으로 정리되어 있는 것

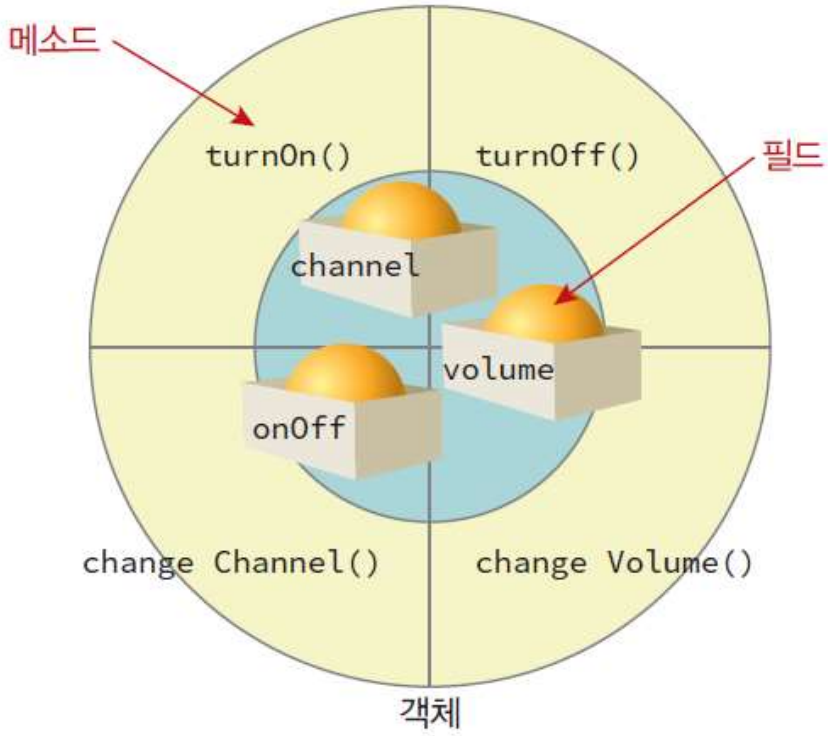

캡슐화 되어 있지 않은
데이터와 코드는 사용하기
어렵겠죠!

캡슐화와 정보 은닉

- 정보 은닉(information hiding)은 객체를 캡슐로 싸서 객체의 내부를 보호하는 하는 것이다. 즉 객체의 실제 구현 내용을 외부에 감추는 것이다.



캡슐화와 정보 은닉

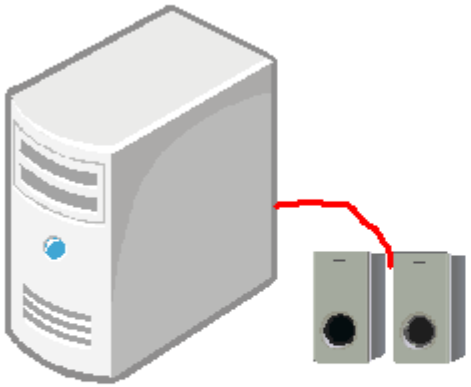

메소드
turnOn()
turnOff()
필드
channel
volume
onOff
change Channel()
change Volume()
객체


보통은 데이터들은 공개되지 않고 몇 개의 메소드만이 외부로 공개됩니다.

업그레이드가 쉽다.

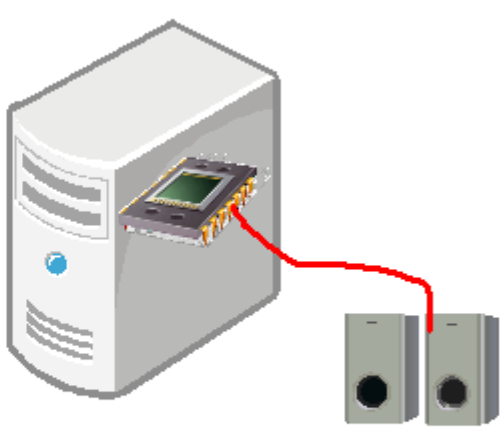
- 라이브러리가 업그레이드되면 쉽게 바꿀 수 있다.
- 정보 은닉이 가능하기 때문에 업그레이드 가능

O



만약 외부의 표준 오디오 단자를 이용하였으면 내부의 사운드 카드를 변경할 수 있다.

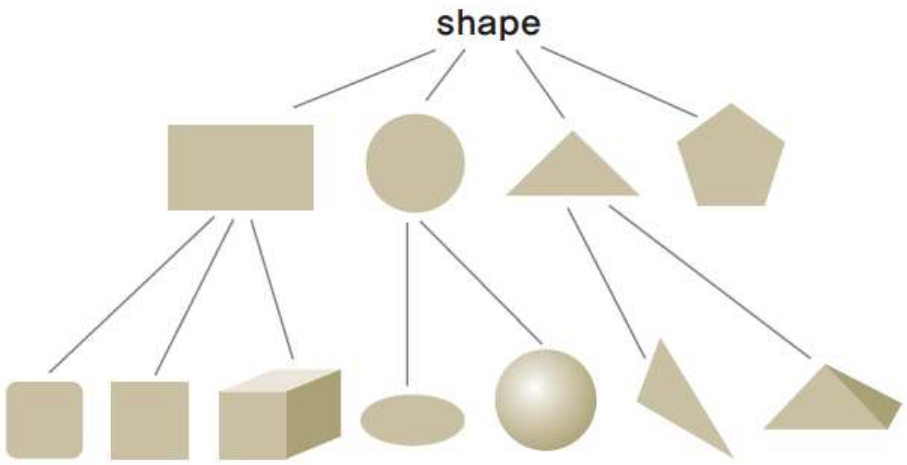
X



만약 내부의 오디오 제어칩의 단자에 연결하였으면 내부의 사운드 카드를 변경할 수 없다.

상속

- 상속(inheritance): 이미 작성된 클래스(부모 클래스)를 이어받아서 새로운 클래스(자식 클래스)를 생성하는 기법
- 기존의 코드를 재활용하기 위한 기법

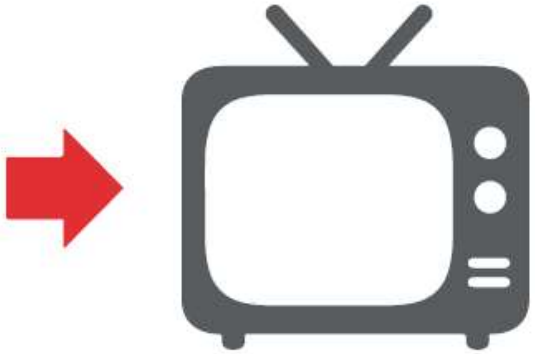


다형성

- 하나의 이름(방법)으로 많은 상황에 대처하는 기법
- 개념적으로 동일한 작업을 하는 멤버 함수들에 똑같은 이름을 부여할 수 있으므로 코드가 더 간단해진다

추상화

실제 객체



추상화된 객체

클래스

- 클래스(class): 객체를 만드는 설계도
- 클래스로부터 만들어지는 각각의 객체를 특별히 그 클래스의 **인스턴스(instance)**라고도 한다.

클래스의 구조

형식

```
class 클래스이름 {
    자료형 필드1;
    자료형 필드2;
    ....

    반환형 메소드1()      ... }
    반환형 메소드2()      ... }
    ...
}
```

메소드 정의
객체의 동작을 나타낸다.

필드 정의
객체의 속성을 나타낸다.

클래스의 예: 박스

- 텔레비전

Television.java

```
01 public class Television {
02     int channel;        // 채널 번호
03     int volume;         // 볼륨
04     boolean onOff;      // 전원 상태
05 }
```

필드 정의
객체의 속성을 나타낸다.

예제: 객체 생성하기

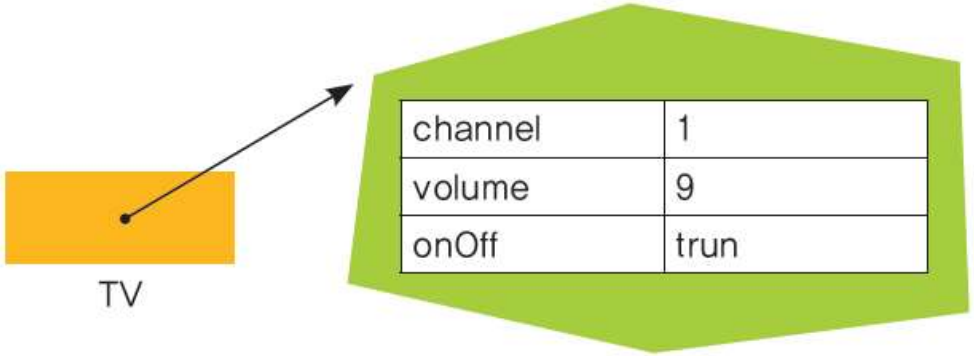
TelevisionTest.java

```
01 public class TelevisionTest {
02     public static void main(String[] args) {
03         Television  tv = new Television();
04         tv.channel = 7;
05         tv.volume = 9;
06         tv.onOff = true;
07         System.out.println("텔레비전의 채널은 " + tv.channel + "이고 볼륨은 "
08             + tv.volume + "입니다.");
09     }
10 }
```

객체를 생성한다.

객체의 멤버에 접근할 때는 멤버 연산자(.)를 사용한다.


```
텔레비전의 채널은 7이고 볼륨은 9입니다.
```


TV
channel
1
volume
9
onOff
trun


당분간 소스 코드 다음
에는 소스 코드를 설명하는
그림이 등장할 것입니다!

변수의 종류

- 기초 변수(primitive variable)에는 실제 데이터값이 저장된다.
- 참조 변수(reference variable)는 참조 변수는 객체를 참조할 때 사용되는 변수로서 여기에는 객체의 참조값이 저장된다.

객체의 필드와 메소드 사용

- 도트(.) 연산자 사용!

여러 개의 객체 생성하기

Television.java

```
01 public class Television {
02     int channel;        // 채널 번호
03     int volume;         // 볼륨
04     boolean onOff;      // 전원 상태
05 }
```

TelevisionTest.java

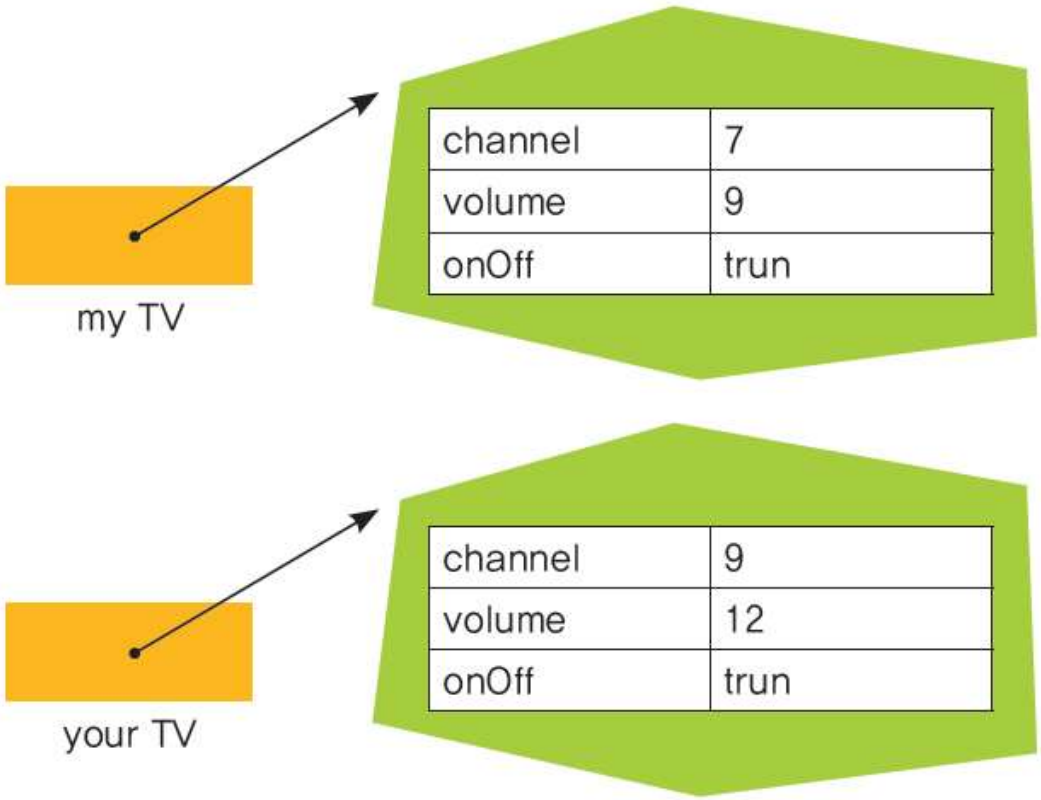
```
01 public class TelevisionTest {
02     public static void main(String[] args) {
03         Television  myTv = new Television();
04         myTv.channel = 7;
05         myTv.volume = 10;
06         myTv.onOff = true;
07
```

myTv와 yourTv는 별개의 객체를 가리킨다.

```
08        Television  yourTv = new Television();
09        yourTv.channel = 9;
10        yourTv.volume = 12;
11        yourTv.onOff = true;
12        System.out.println("나의 텔레비전의 채널은 " + myTv.channel +
13           "이고 볼륨은 " + myTv.volume + "입니다.");
14        System.out.println("너의 텔레비전의 채널은 " + yourTv.channel +
15           "이고 볼륨은 " + yourTv.volume + "입니다.");
16     }
17  }
```


```
나의 텔레비전의 채널은 7이고 볼륨은 10입니다.
너의 텔레비전의 채널은 9이고 볼륨은 12입니다.
```

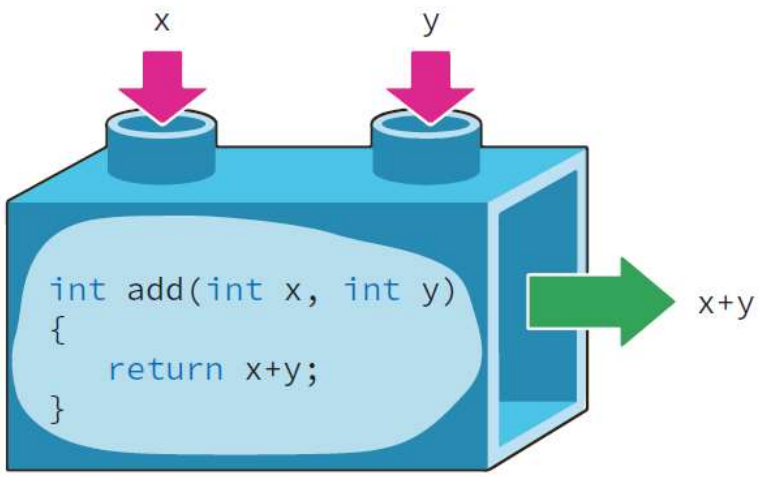
실행 결과


my TV
channel 7
volume 9
onOff trun
your TV
channel 9
volume 12
onOff trun

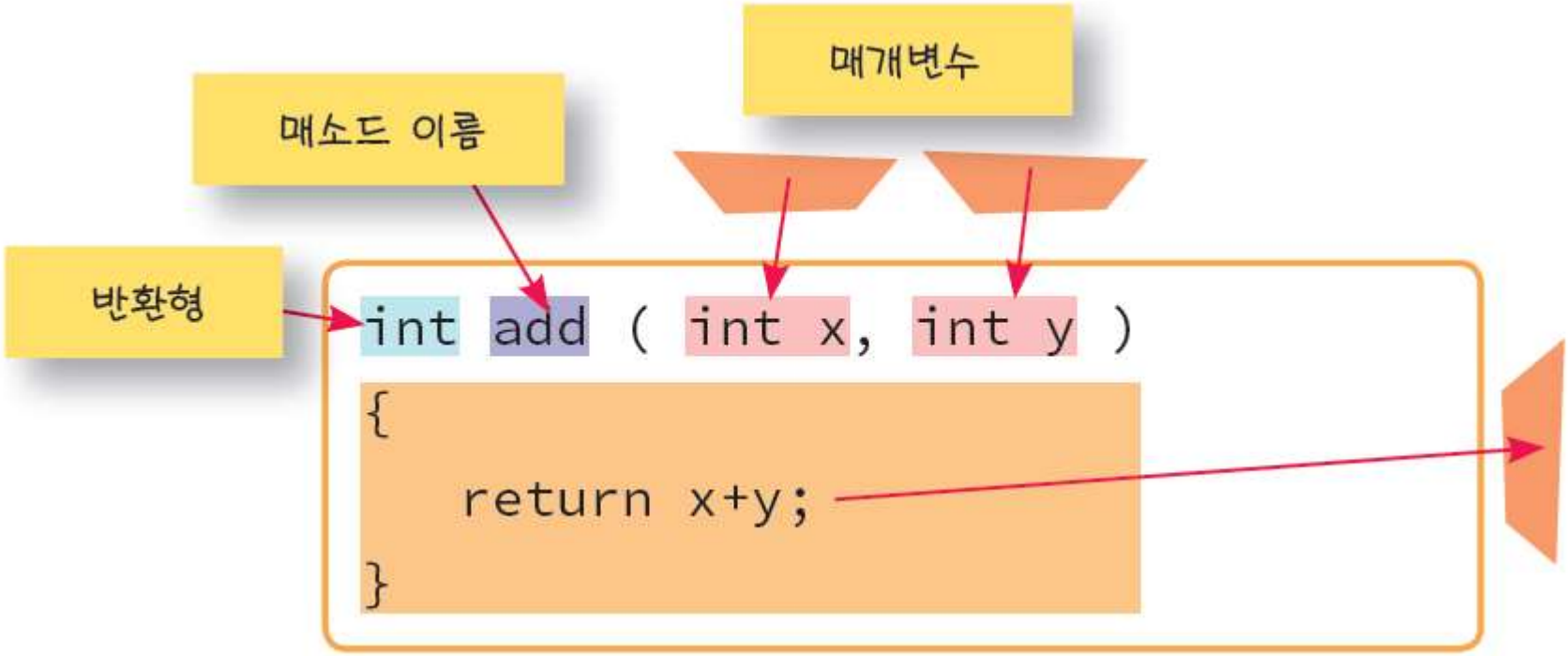

여기서 중요한 것은 각 객체마다 별도의 변수(필드)를 가진다는 점입니다.

메소드

- 메소드는 입력을 받아서 처리를 하고 결과를 반환하는 가상적인 상자와 같다.



메소드의 구조



예 제

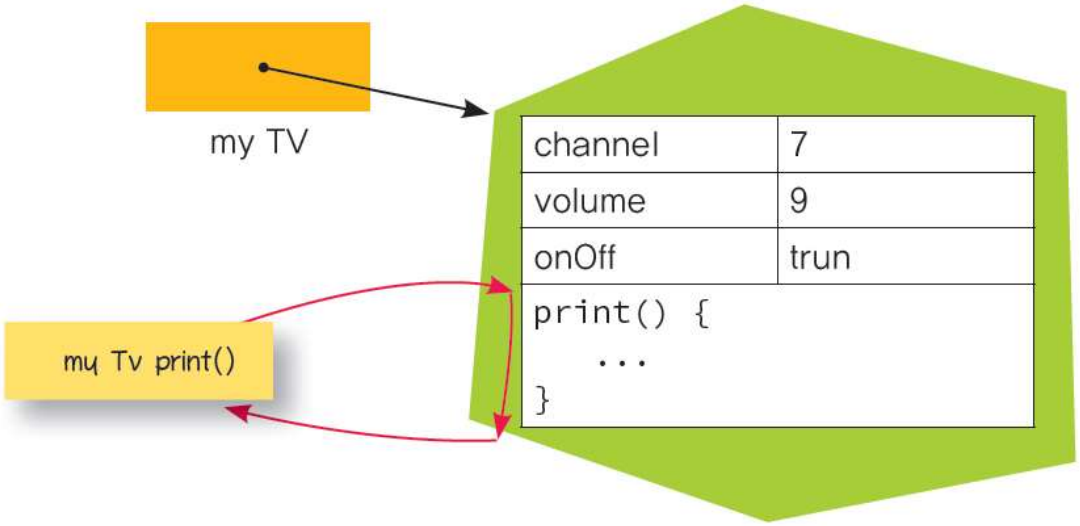
```
public class Television {
        int channel;            // 채널 번호
        int volume;             // 볼륨
        boolean onOff;          // 전원 상태
        void print() {
                System.out.println("채널은 " + channel +
                        "이고 볼륨은 " + volume + "입니다.");
        }
}
```

예 제

```
public class TelevisionTest {
        public static void main(String[] args) {
                Television myTv = new Television();
                myTv.channel = 7;
                myTv.volume = 9;
                myTv.onOff = true;
                myTv.print();
                Television yourTv = new Television();
                yourTv.channel = 9;
                yourTv.volume = 12;
                yourTv.onOff = true;
                yourTv.print();

        }
}
```


```
채널은 7이고 볼륨은 10입니다.
채널은 9이고 볼륨은 12입니다.
```

예제 설명



myTv.print() 문장이 실행되면 myTv 안의 print() 메소드가 실행되고 실행이 끝나면 myTv.print() 문장으로 되돌아옵니다.

메소드의 종료

- return 을 사용한다.

```
void myMethod() {
        for( int i=0; i<10; i++ ) {
                if( i == 7 )
                        return;
        }
}
```

메소드의 반환값

형식

```
return 반환값;
```

예 제

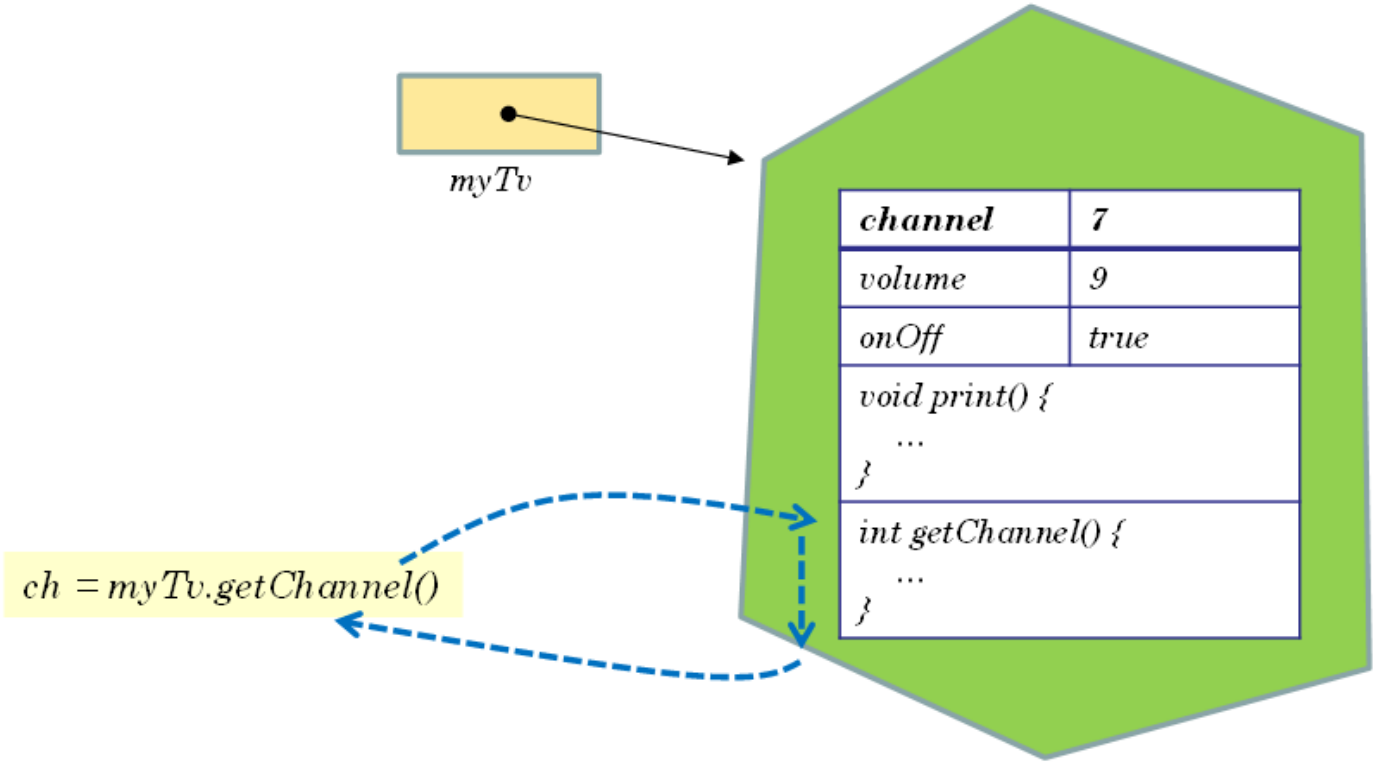
```
public class T elevision {
        int channel; // 채널 번호
        int volume; // 볼륨
        boolean onOff; // 전원 상태
        void print() {
                System.out.println("채널은 " + channel + "이고 볼륨은 " + volume +
"입니다.");
        }
        int getChannel() {
                return channel;
        }
}
```

예 제

```
public class TelevisionTest {
        public static void main(String[] args) {
                Television myTv = new Television();
                myTv.channel = 7;
                myTv.volume = 9;
                myTv.onOff = true;
                int ch = myTv.getChannel();
                System.out.println("현재 채널은 " + ch + "입니다.");
        }
}
```

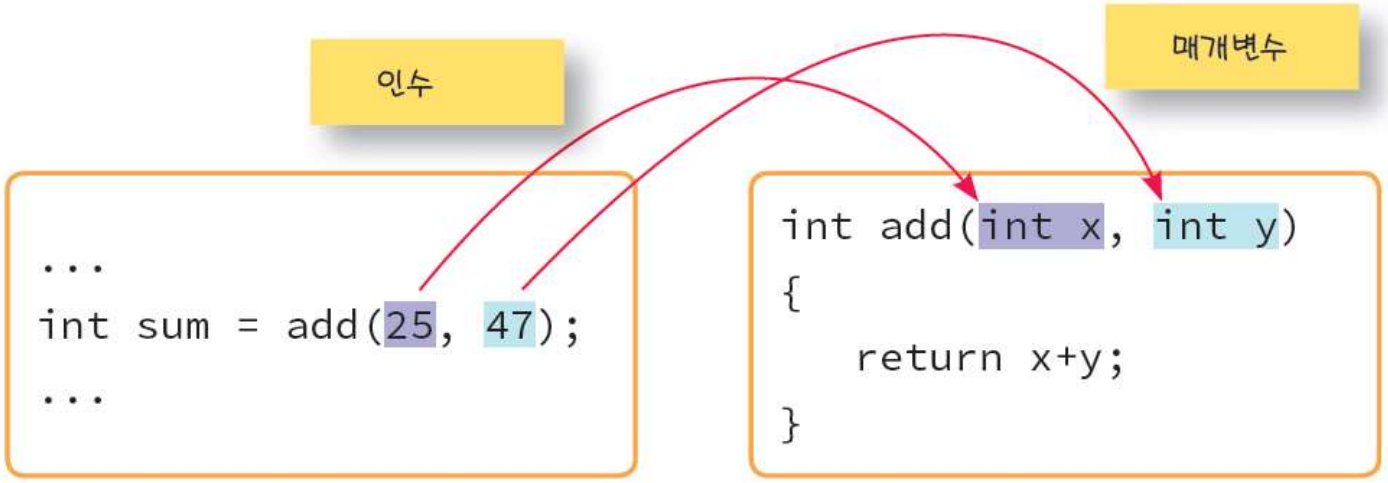

```
현재 채널은 7입니다.
```

예제 설명


myTv
channel
7
volume
9
onOff
true
void print() {
…
}
int getChannel() {
…
}
ch = myTv.getChannel()

인수와 매개 변수

- 메소드 호출시 전달하는 값을 인수(argument)
- 메소드에서 값을 받을 때 사용하는 변수를 매개 변수(parameter)



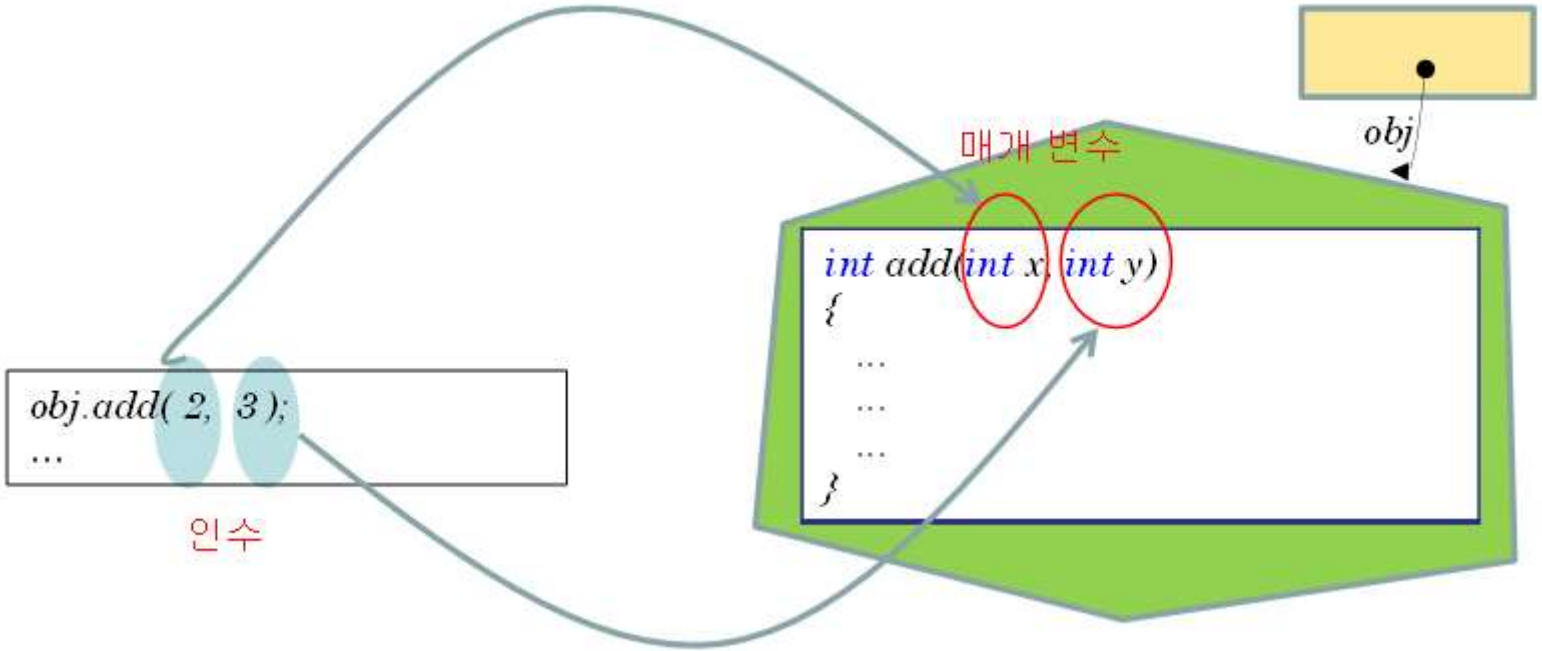
예 제

```
public class Math {
        int add(int x, int y) {
                return x + y;
        }
}
```

```
public class MathTest {
        public static void main(String[] args) {
                int sum;
                Math obj = new Math();
                sum = obj.add(2, 3);
                System.out.println("2와 3의 합은 " + sum);
                sum = obj.add(7, 8);
                System.out.println("7와 8의 합은 " + sum);
        }
}
```


```
2와 3의 합은 5
7와 8의 합은 15
```

예제 설명



예제

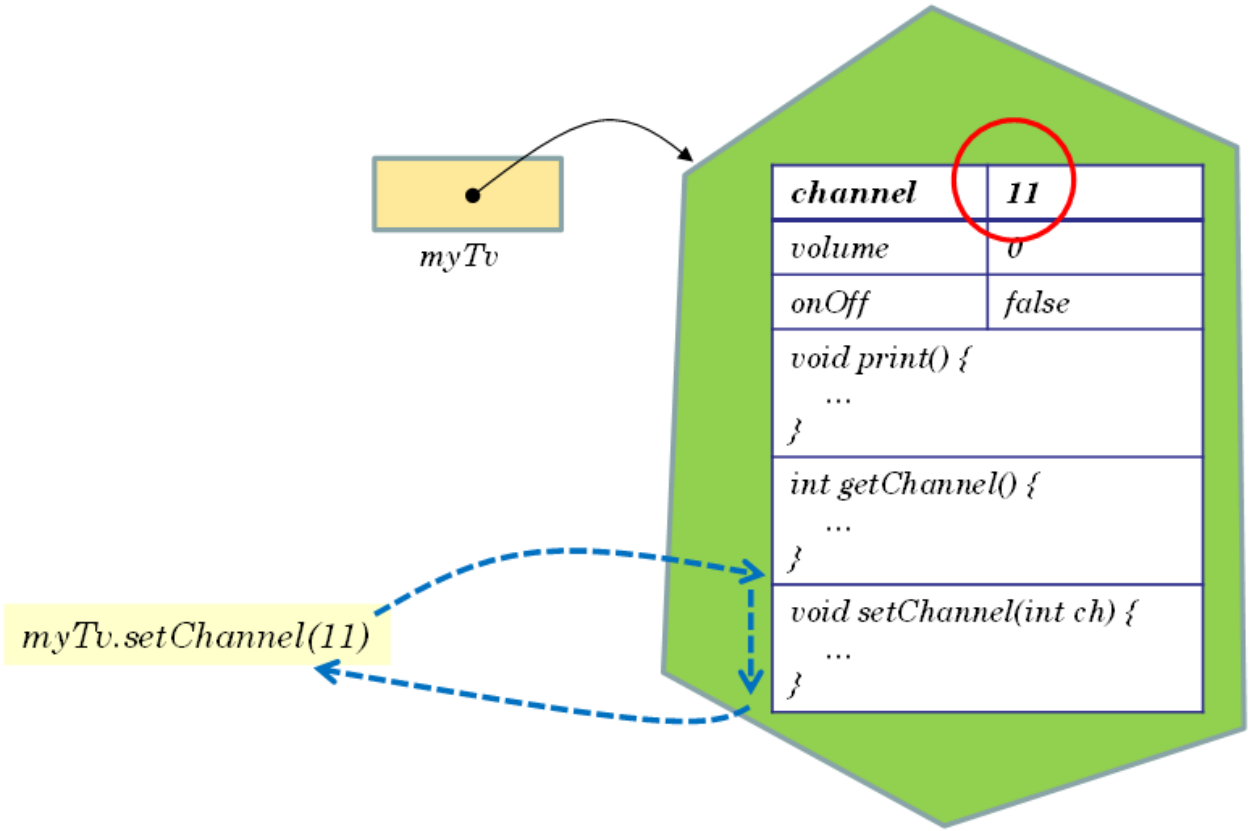
```
public class Television {
        int channel; // 채널 번호
        int volume; // 볼륨
        boolean onOff; // 전원 상태
        void print() {
                System.out.println("채널은 " + channel + "이고 볼륨은 " + volume +
"입니다.");
        }
        int getChannel() {
                return channel;
        }
        void setChannel(int ch) {
                channel = ch;
        }
}
```

예 제

```
public class TelevisionTest {
        public static void main(String[] args) {
                Television myTv = new Television();
                myTv.setChannel(11);
                int ch = myTv.getChannel();
                System.out.println("현재 채널은 " + ch + "입니다.");
        }
}
```

현재 채널은 11입니다.

예제 설명


myTv
channel
11
volume
0
onOff
false
void print() {
…
}
int getChannel() {
…
}
void setChannel(int ch) {
…
}
myTv.setChannel(11)

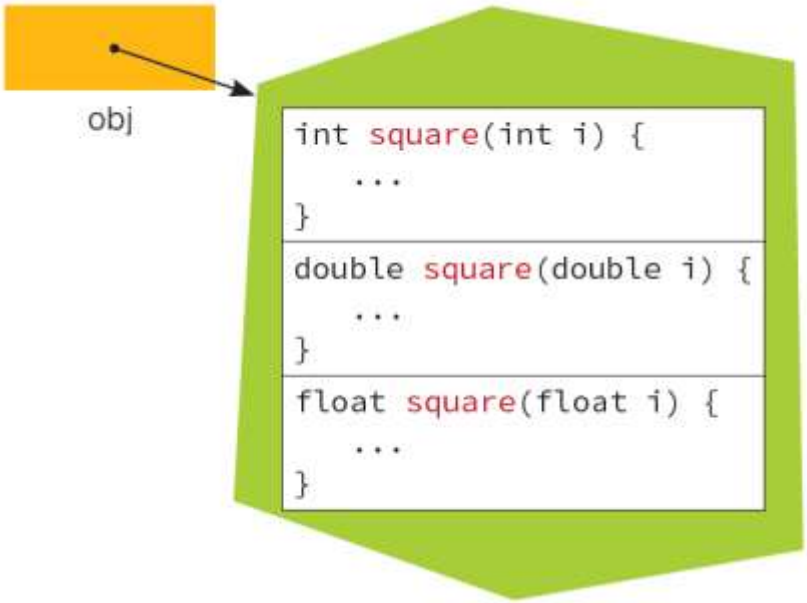
예 제

```
public class CarTest {
        public static void main(String[] args) {
                Car myCar = new Car();
                myCar.changeGear(1);
                myCar.speedUp();
                System.out.println(myCar);
        }
}
```


```
Car [color=null, speed=10, gear=1]
```

메소드 오버로딩

- 자바에서는 같은 이름의 메소드가 여러 개 존재할 수 있다. 이것을 **메소드 오버로딩(method overloading)**이라고 한다.



예 제

```
public class MyMath {
        // 정수값을 제곱하는 메소드
        int square(int i) {
                return i * i;
        }
        // 실수값을 제곱하는 메소드
        double square(double i) {
                return i * i;
        }
}
```

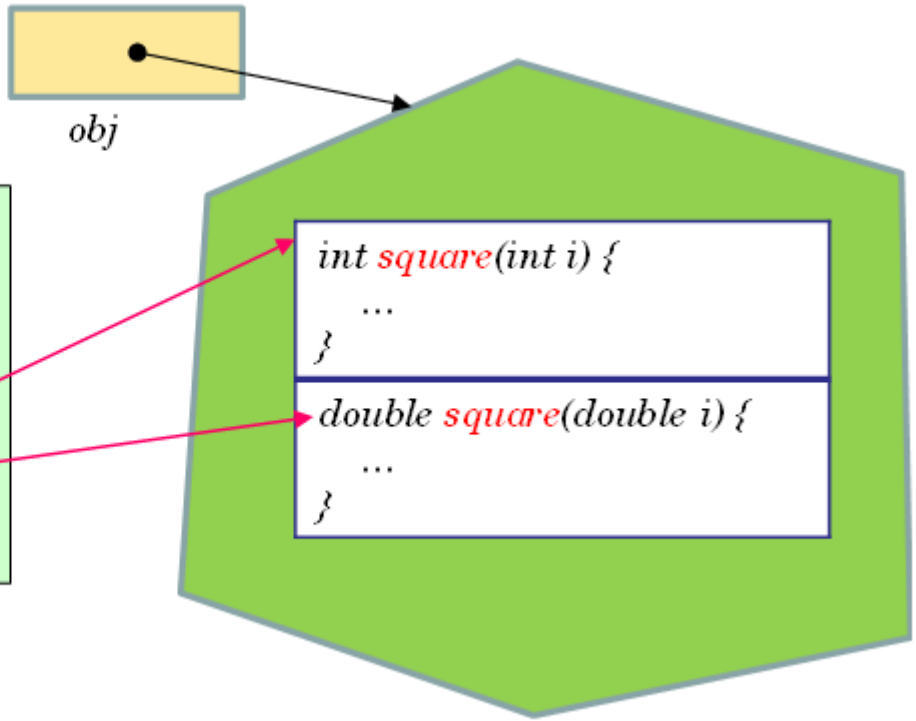
예 제

```
public class MyMathTest {
        public static void main(String args[]) {
                MyMath obj = new MyMath();
                System.out.println(obj.square(10));
                System.out.println(obj.square(3.14));
        }
}
```


```
100
9.8596
```

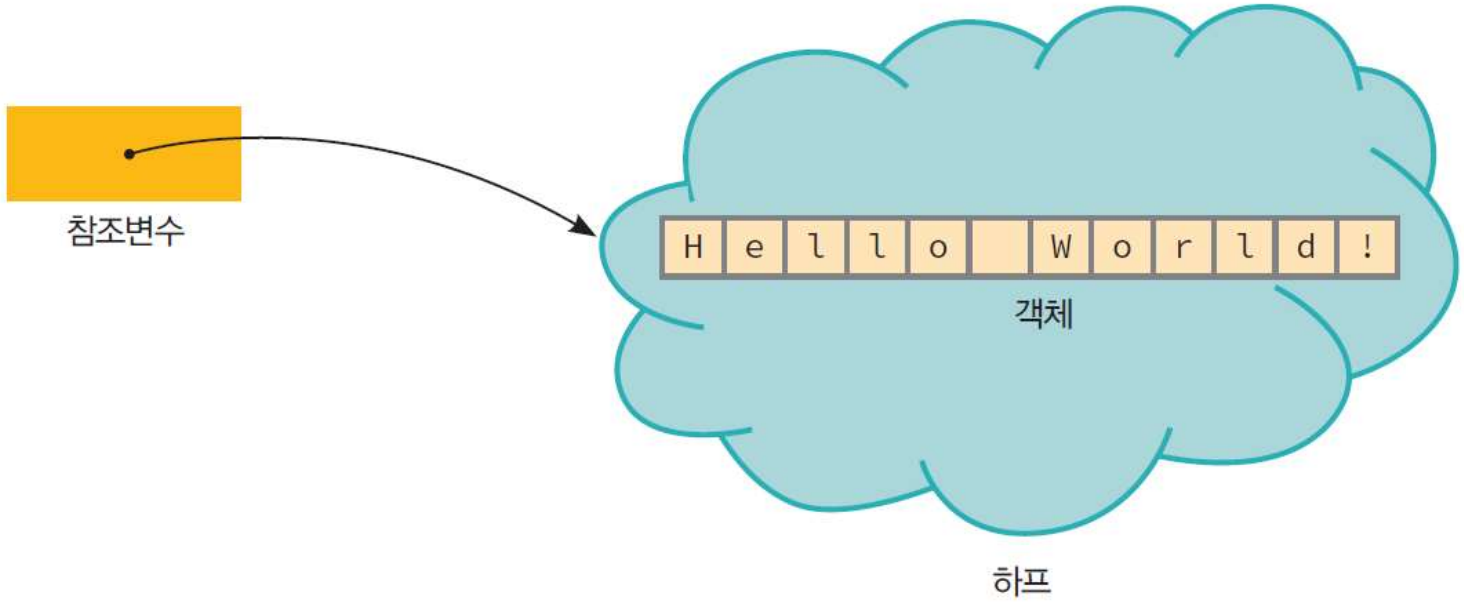
예제 설명

```
MyMath obj = new MyMath();

System.out.println(obj.square(10));
System.out.println(obj.square(3.14));
```



STRING 클래스의 객체 생성

- String s = **new** String("Hello World!"); // 선언과 동시에 초기화



STRING 클래스의 메소드

반환형	메소드 요약
char	charAt(int index) 지정된 인덱스에 있는 문자를 반환한다.
int	compareTo(String anotherString) 사전적 순서로 문자열을 비교한다. 앞에 있으면 −1, 같으면 0, 뒤에 있으면 1이 반환된다.
String	concat(String str) 주어진 문자열을 현재의 문자열 뒤에 붙인다.
boolean	equals(Object anObject) 주어진 객체와 현재의 문자열을 비교한다.
boolean	equalsIgnoreCase(String anotherString) 대소문자를 무시하고 비교한다.
boolean	isEmpty() length()가 0이면 true를 반환한다.
int	length() 현재 문자열의 길이를 반환한다.

STRING 클래스 사용하기

```
public class StringTest
{
  public static void main (String[] args)
  {
    String proverb = "A barking dog";                    // new 연산자 생략
    String s1, s2, s3, s4;          // 참조 변수로서 메소드에서 반환된 참조값을 받는다.

    System.out.println ("문자열의 길이 =" + proverb.length());

    s1 = proverb.concat (" never Bites!");      // 문자열 결합
    s2 = proverb.replace ('B', 'b');            // 문자 교환
    s3 = proverb.substring (2, 5);              // 부분 문자열 추출
    s4 = proverb.toUpperCase();                 // 대문자로 변환

    System.out.println(s1);
    System.out.println(s2);
    System.out.println(s3);
    System.out.println(s4);
  }
}
```

예 제


```
문자열의 길이 =13
A barking dog never Bites!
A barking dog
bar
A BARKING DOG
```

문자열의 결합

- + 연산자를 사용한다.

```
String subject = "Money";
String other = " has no value if it is not used";
String sentence = subject + other;                    // "Money has no value if it is not used"
```

수치값-> 문자열

- 자바에서는 문자열과 기초 자료형 변수를 결합하게 되면 자동적으로 기초 자료형을 문자열로 변환한다.

```
int x = 20;
System.out.println("결과값은 " + x);          // ”결과값은 20” 이 출력된다.
String answer = "The answer is " + 100;      // “The answer is 100"
```

문자열->수치값

- 즉 문자열 "123"을 숫자 123으로 변환하려면 어떻게 하여야 하는가? 자바에는 이것을 전문으로 해주는 클래스가 있다. 바로 랩퍼 클래스인 Integer 클래스이다.

기초 자료형	랩퍼 클래스
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
char	Character
boolean	Boolean
void	Void

예제

- 문자열을 기초 자료형으로 변환하려면 각 랩퍼 클래스의 parseXXX() 메소드를 사용한다.

```
int i = Integer.parseInt("123");                    // 변수 i에 정수 123이 저장된다.
double d = Double.parseDouble("3.141592"); // 변수 d에 실수 3.141592가 저장된다.
```

어서와 JAVA는 처음이지!

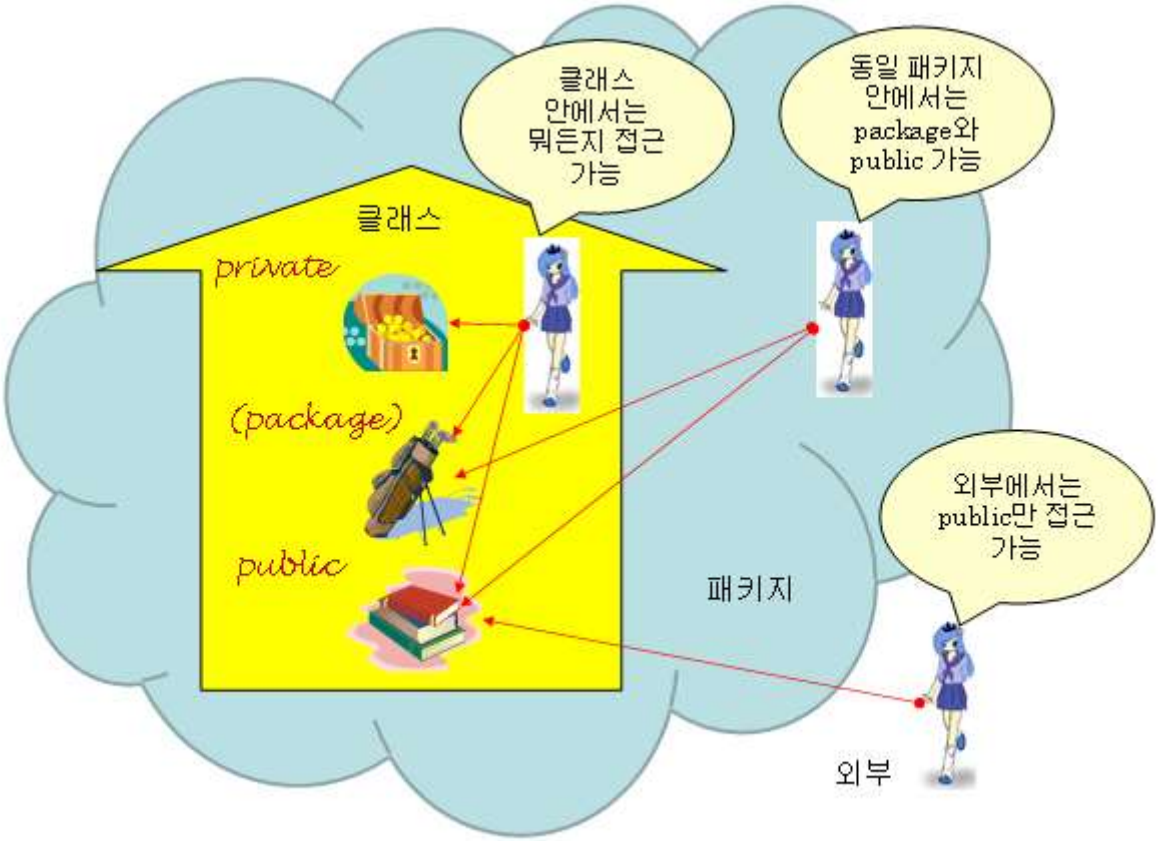
제6장 클래스, 메소드 심층연구

접근 제어

- 클래스 안에 변수나 메소드들을 누구나 사용할 수 있게 하면 어떻게 될까? -> 많은 문제가 발생할 것이다.

접근 제어

- 접근 제어(access control): 다른 클래스가 특정한 필드나 메소드에 접근하는 것을 제어하는 것



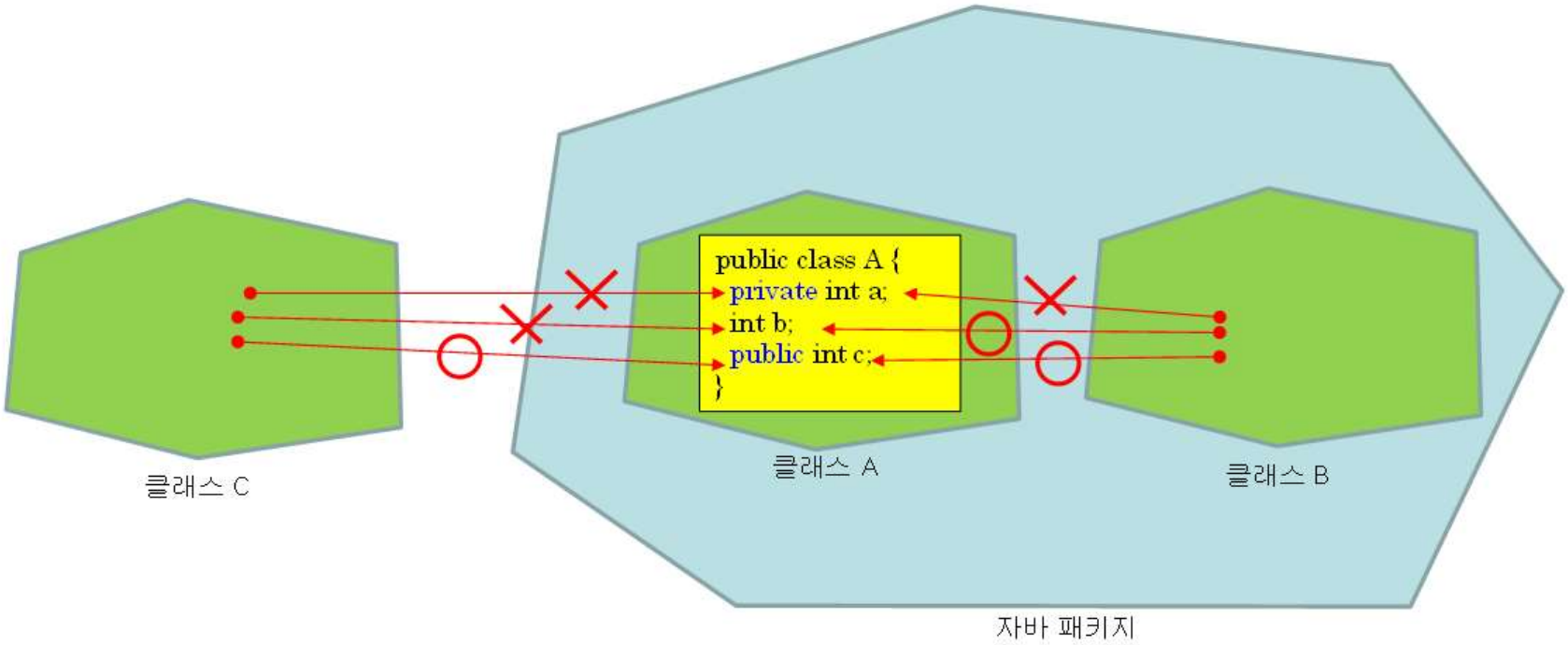
멤버 수준에서의 접근 제어

접근 지정자	클래스	패키지	자식 클래스	전체 세계
public	O	O	O	O
protected	O	O	O	X
없음	O	O	X	X
private	O	X	X	X

예 제

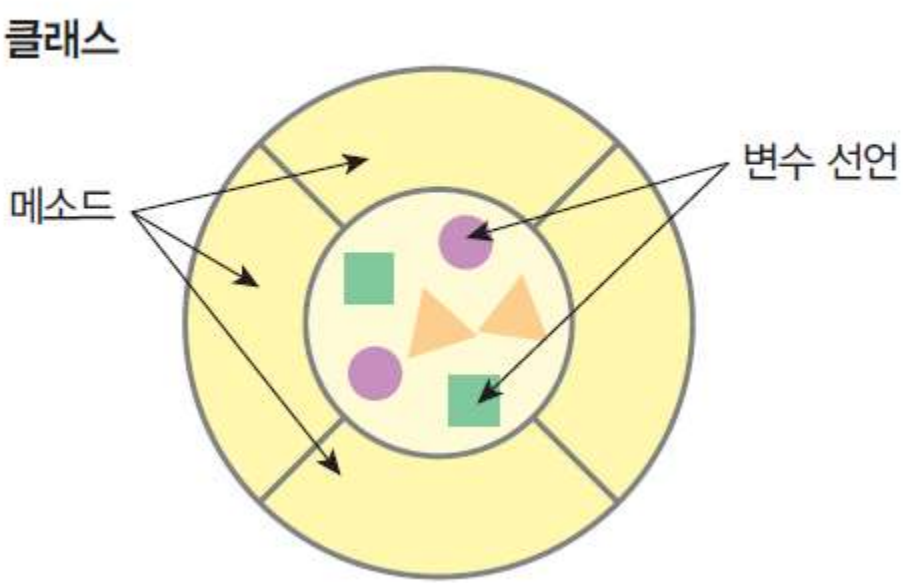
```
class A {
        private int a;          // 전용
        int b;                  // 디폴트
        public int c;           // 공용
}

public class Test {
        public static void main(String args[]) {
                A obj = new A();        // 객체 생성
                obj.a = 10;             // 전용 멤버는 다른 클래스에서는 접근 안 됨
                obj.b = 20;             // 디폴트 멤버는 접근할 수 있음
                obj.c = 30;             // 공용 멤버는 접근할 수 있음
        }
}
```

예 제


public class A {
private int a;
int b;
public int c;
}
클래스 C
클래스 A
클래스 B
자바 패키지

정보은닉

- 정보 은닉이란 구현의 세부 사항을 클래스 안에 감추는 것이다



예 제

```
public class AccountTest {
        public static void main(String[] args) {
                Account obj = new Account();
                obj.setName("Tom");
                obj.setBalance(100000);
                System.out.println("이름은 " + obj.getName() + " 통장 잔고는
"
                                + obj.getBalance() + "입니다.");
        }
}
```

이름은 Tom 통장 잔고는 100000입니다.

생성자

- 생성자(contructor): 객체가 생성될 때에 필드에게 초기값을 제공하고 필요한 초기화 절차를 실행하는 메소드

생성자 정의

형식

```
public class Car {
    Car() {
        ...
    }
}
```

클래스 이름과 동일한 메소드가 바로 생성자입니다. 여기서 객체의 초기화를 담당합니다.

생성자의 예

```
public class MyCounter {
        int counter;
        MyCounter() {
                counter = 1;
        }
}
```

생성자의 예

```
public class MyCounterTest {

        public static void main(String args[]) {
                MyCounter obj1 = new MyCounter();
                MyCounter obj2 = new MyCounter();
                System.out.println("객체 1의 counter = " + obj1.counter);
                System.out.println("객체 2의 counter = " + obj2.counter);
        }
}
```


```
객체 1의 counter = 1
객체 2의 counter = 1
```

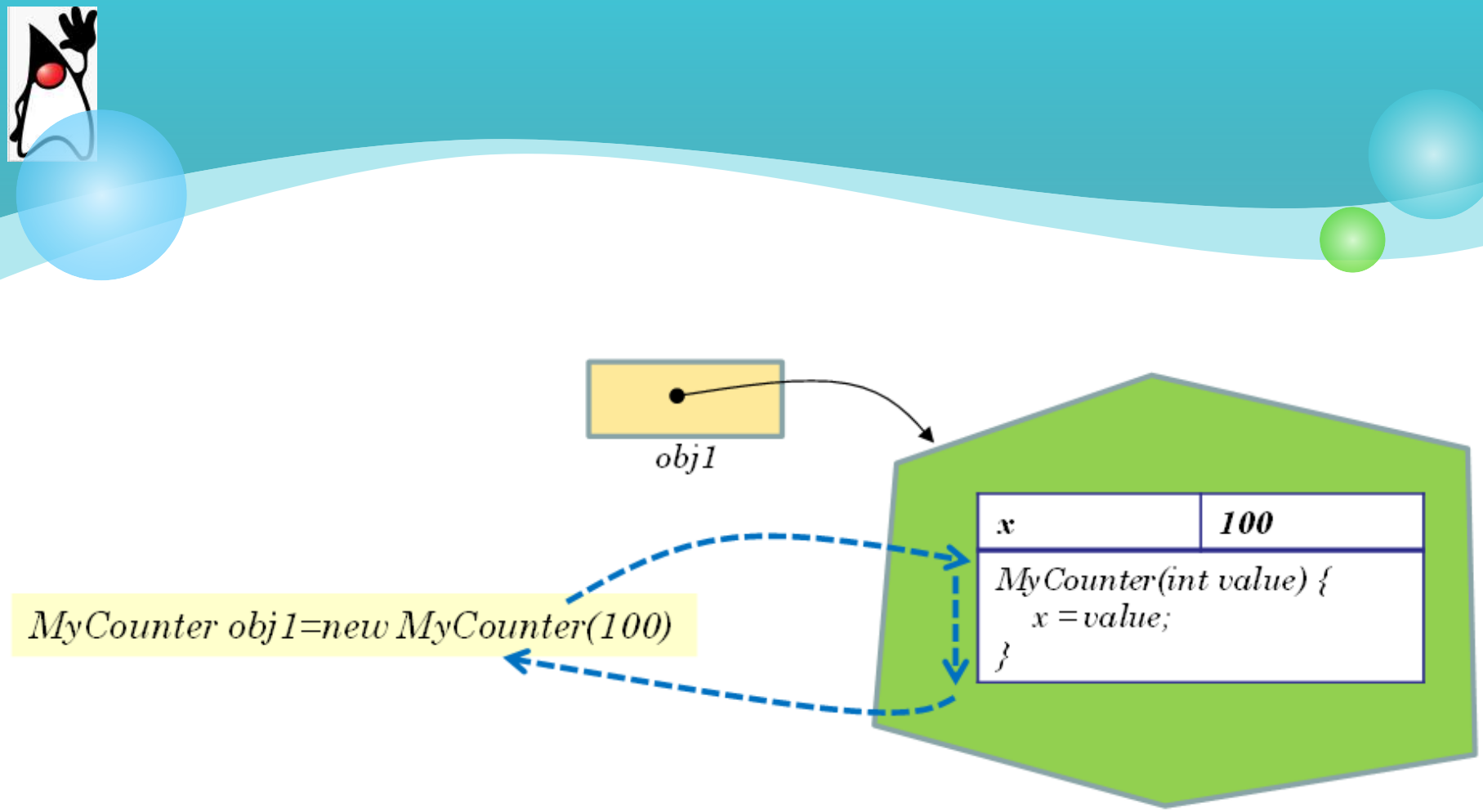
매개변수를 가지는 생성자

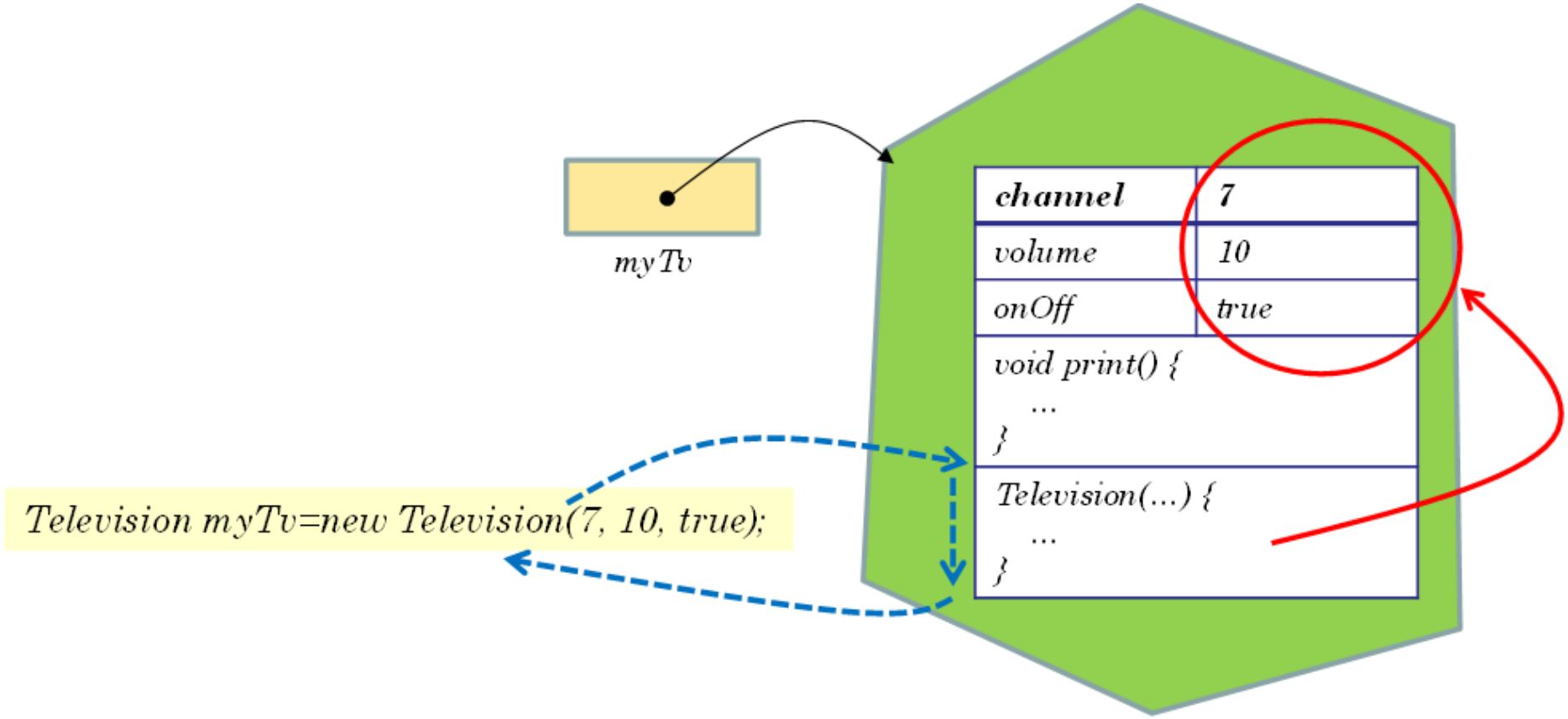
```
class MyCounter {
        int counter;

        MyCounter(int value) {
                counter = value;
        }
}
```

생성자의 예

```
public class MyCounterTest {
        public static void main(String args[]) {
                MyCounter obj1 = new MyCounter(100);
                MyCounter obj2 = new MyCounter(200);

                System.out.println("객체 1의 counter = " + obj1.counter);
                System.out.println("객체 2의 counter = " + obj2.counter);
        }
}
```


```
객체 1의 counter = 100
객체 2의 counter = 200
```


obj1
x
100
MyCounter(int value) {
x =value;
}
MyCounter obj1=new MyCounter(100)


myTv
channel
7
volume
10
onOff
true
void print() {
...
}
Television(...) {
...
}
Television myTv=new Television(7, 10, true);

생성자 오버로딩

- 메소드처럼 생성자도 오버로딩될 수 있다.

예제

```
public class Student {
        private int number;
        private String name;
        private int age;

        Student() {
                number = 100;
                name = "New Student";
                age = 18;
        }

        Student(int number, String name, int age) {
                this.number = number;
                this.name = name;
                this.age = age;
        }

        @Override
        public String toString() {
                return "Student [number=" + number + ", name=" + name + ",
age=" + age + "]";
        }
}
```

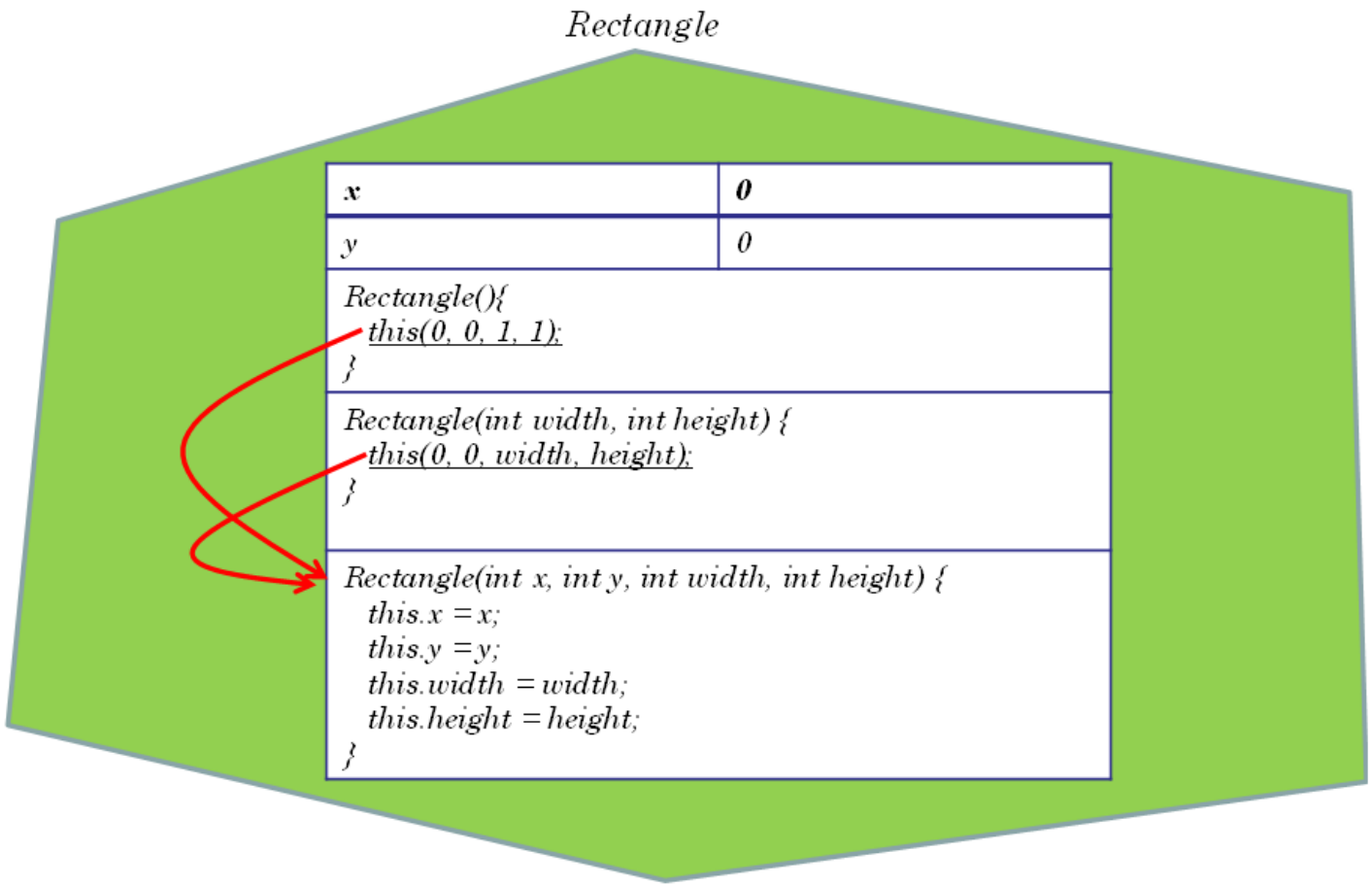
THIS로 현재 객체 나타내기

- 메소드나 생성자에서 this는 현재 객체를 나타낸다.

```
public class Point {
    public int x = 0;
    public int y = 0;

    // 생성자
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

THIS()

```
public class Rectangle {
        private int x, y;
        private int width, height;

        Rectangle() {
                this(0, 0, 1, 1);
        }

        Rectangle(int width, int height) {
                this(0, 0, width, height);
        }

        Rectangle(int x, int y, int width, int height) {
                this.x = x;
                this.y = y;
                this.width = width;
                this.height = height;
        }
        // ...
}
```


Rectangle
x
0
y
0
Rectangle(){
this(0, 0, 1, 1);
}
Rectangle(int width, int height) {
this(0, 0, width, height);
}
Rectangle(int x, int y, int width, int height) {
this.x = x;
this.y = y;
this.width = width;
this.height = height;
}

필드 초기화 방법

- 필드 선언시 초기화

```
public class Hotel {
   public int capacity = 10;                    // 10으로 초기화한다.
     private boolean full = false;                    // false로
초기화한다.
          ...
}
```

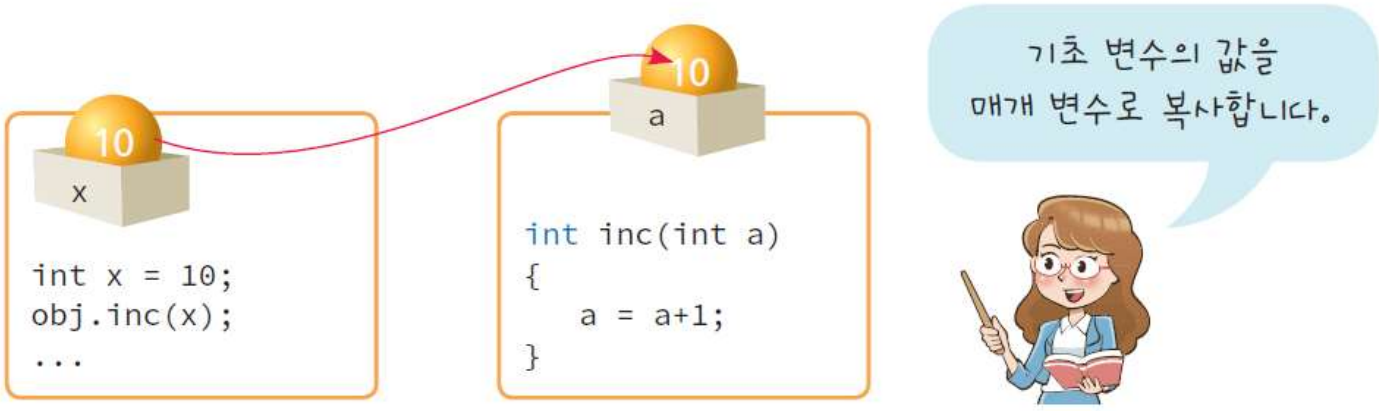
필드 초기화 방법

- 인스턴스 초기화 블록(instance initializer block)

```
public class Car {
        int speed;
        Car() {
                System.out.println("속도는 " + speed);
        }

        {
                speed = 100;
        }
        public static void main(String args[]) {
                Car c1 = new Car();
                Car c2 = new Car();
        }
}
```

메소드로 기초형 변수가 전달되는 경우

- 기초형 변수가 전달되는 경우



예제

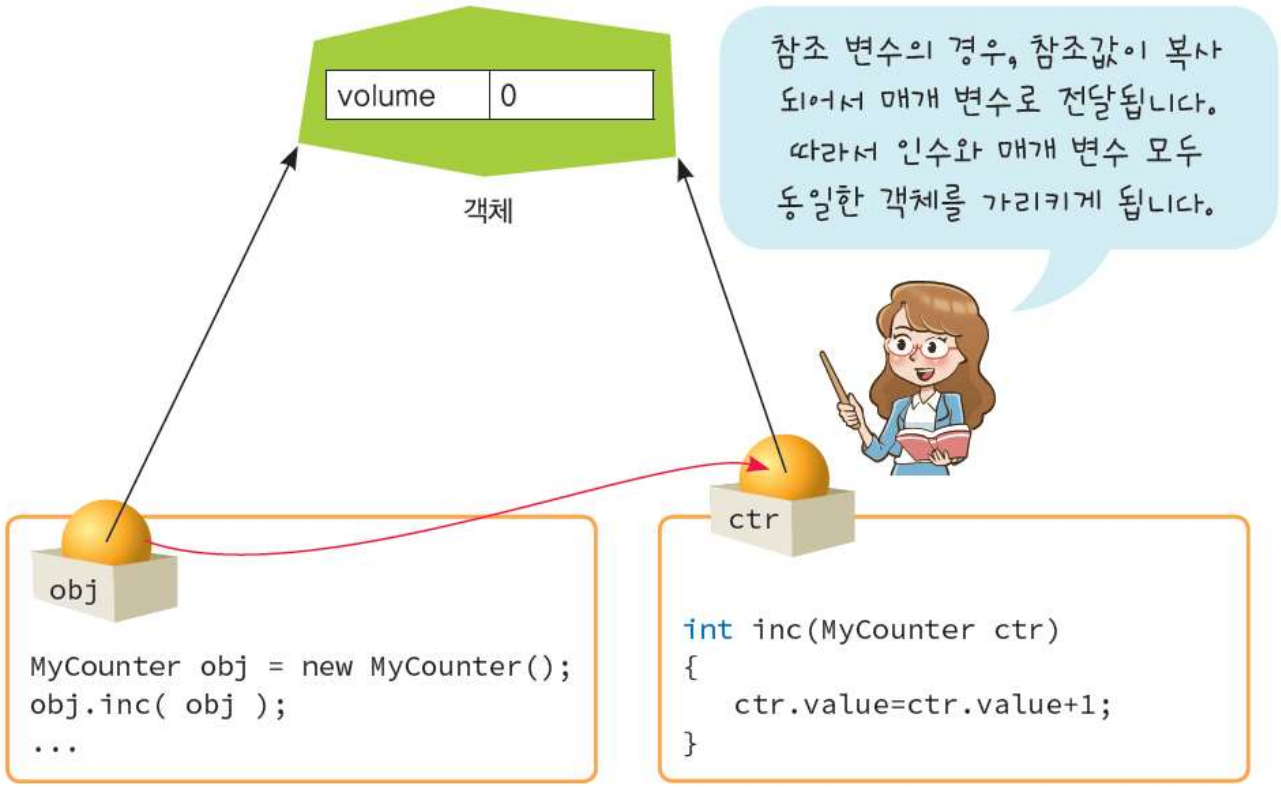
```
public class MyCounter {
        int value;
        void inc(int a) {
                a = a + 1;
        }
}
```

```
public class MyCounterTest1 {
        public static void main(String args[]) {
                MyCounter obj = new MyCounter();
                int x = 10;
                obj.inc(x);
                System.out.println("x = " + x);
        }
}
```


```
x = 10
```

메소드로 객체가 전달되는 경우

- 객체를 메소드로 전달하게 되면 객체가 복사되어 전달되는 것이 아니고 참조 변수의 값이 복사되어서 전달된다.



예제

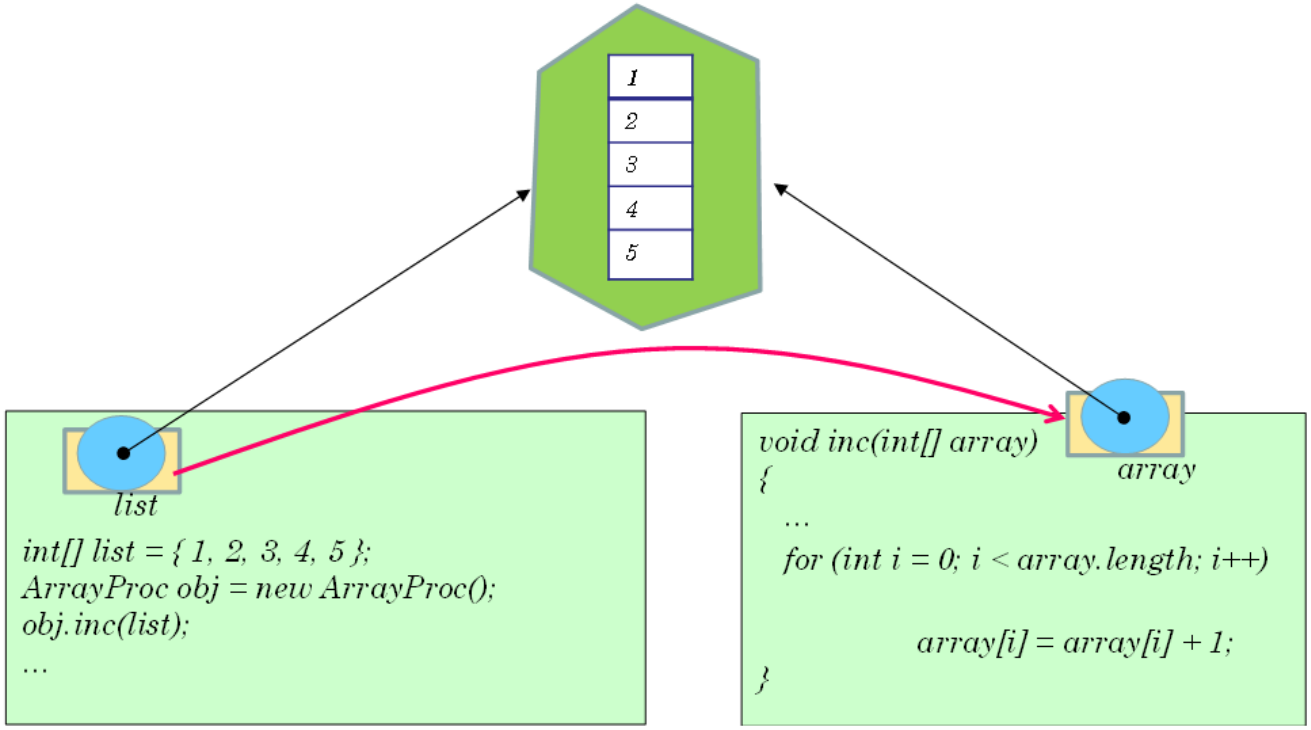
```
class MyCounter {
        int value = 0;
        void inc(MyCounter ctr) {
                ctr.value = ctr.value + 1;
        }
}
```

```
public class MyCounterTest2 {
        public static void main(String args[]) {
                MyCounter obj = new MyCounter();

                System.out.println("obj.value = " + obj.value);
                obj.inc(obj);
                System.out.println("obj.value = " + obj.value);
        }
}
```


```
obj.value = 0
obj.value = 1
```

메소드로 배열이 전달되는 경우

- 배열도 객체이기 때문에 배열을 전달하는 것은 배열 참조 변수를 복사하는 것이다.



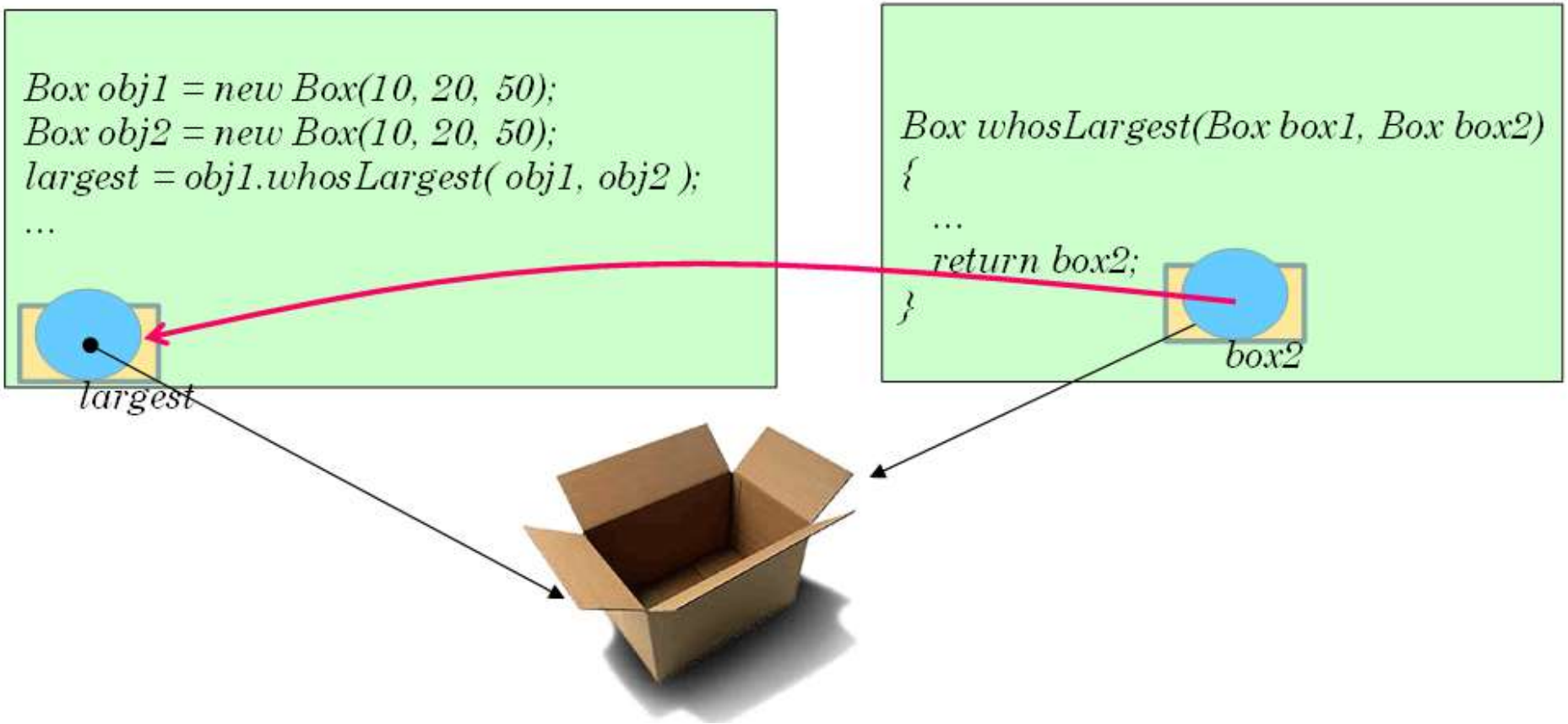
메소드에서 객체 반환하기

```
public class Box {
        int width, length, height;
        int volume;

        Box(int w, int l, int h) {
                width = w;
                length = l;
                height = h;
                volume = w * l * h;
        }

        Box whosLargest(Box box1, Box box2) {
                if (box1.volume > box2.volume)
                        return box1;
                else
                        return box2;
        }
 }
```

메소드에서 객체 반환하기

```
public class BoxTest {
        public static void main(String args[]) {
                Box obj1 = new Box(10, 20, 50);
                Box obj2 = new Box(10, 30, 30);

                Box largest = obj1.whosLargest(obj1, obj2);
                System.out.println("(" + largest.width + "," +
largest.length + ","
                                + largest.height + ")");
        }
}
```


```
(10,20,50)
```


Box obj1 = new Box(10, 20, 50);
Box obj2 = new Box(10, 20, 50);
largest = obj1.whosLargest(obj1, obj2);
...
largest
Box whosLargest(Box box1, Box box2)
{
...
return box2;
}
box2

obj2

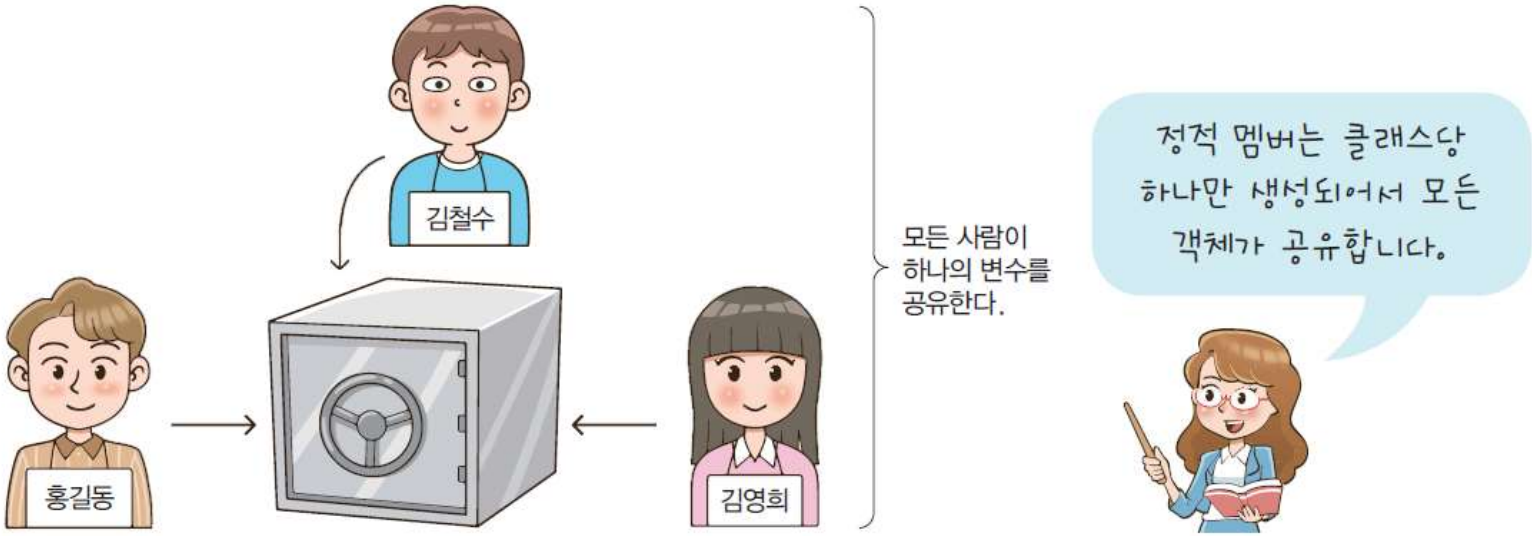
```
Box obj1 = new Box(10, 20, 50);
Box obj2 = new Box(10, 20, 50);
obj1.isSameBox( obj2 );
…
```

obj

```
boolean isSameBox(Box obj)
{
   …
}
```

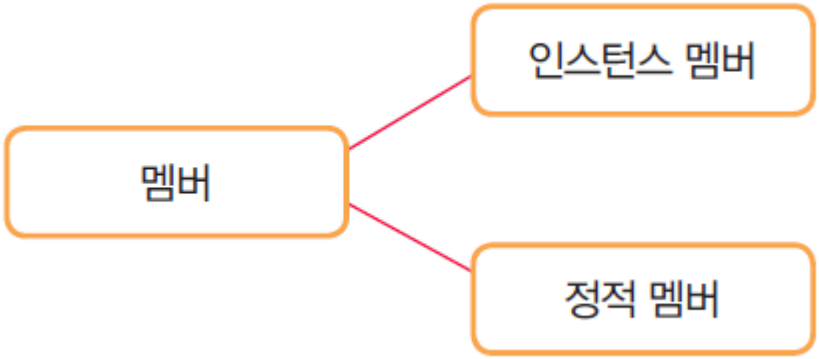
정적 멤버

- 정적 멤버(static member)는 모든 객체를 통틀어서 하나만 생성되고 모든 객체가 이것을 공유하게 된다.



인스턴스 멤버 VS 정적 멤버

- 클래스의 멤버는 인스턴스 멤버와 정적 멤버로 나누어진다.



인스턴스 변수

- 인스턴스마다 별도로 생성되기 때문에 인스턴스 변수(instance variable)라고도 한다.

```
class Television {
    int channel;
    int volume;
    boolean onOff;
}
```

channel	7
volume	9
onOff	trun

객체 A

channel	9
volume	10
onOff	trun

객체 B

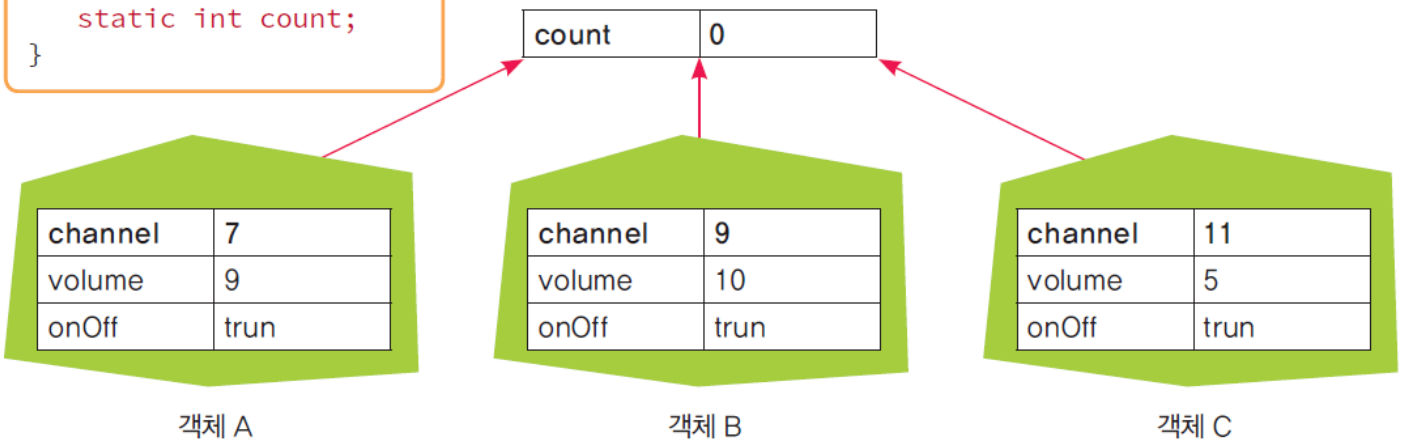
channel	11
volume	5
onOff	trun

객체 C

정적 변수

- 정적 변수는 모든 객체에 공통인 변수이다.
- 정적 변수는 하나의 클래스에 하나만 존재한다

```
class Television {
    int channel;
    int volume;
    boolean onOff;
    static int count;
}
```



정적 변수 예제

```
public class Car {
        private String model;
        private String color;
        private int speed;

        // 자동차의 시리얼 번호
        private int id;
        private static int numbers = 0;

        public Car(String m, String c, int s) {
            model = m;
            color = c;
            speed = s;
            // 자동차의 개수를 증가하고 id에 대입한다.
            id = ++numbers;
        }
}
```

SOLUTION

```
public class CarTest {
    public static void main(String args[]) {
        Car c1 = new Car("S600", "white", 80);          // 첫 번째 생성자 호출
        Car c2 = new Car("E500", "blue", 20);           // 첫 번째 생성자 호출
        int n = Car.numbers;        // 정적 변수
        System.out.println("지금까지 생성된 자동차 수 = " + n);
    }
}
```

지금까지 생성된 자동차 수 = 2

정적 메소드

- 변수와 마찬가지로 메소드도 정적 메소드로 만들 수 있다.
- 정적 메소드는 static 수식자를 메소드 앞에 붙이며 클래스 이름을 통하여 호출되어야 한다.

- (예) double value = Math.sqrt(9.0);

정적 변수 예제

```
public class Car {
    private String model;
    private String color;
    private int speed;

    // 자동차의 시리얼 번호
    private int id;
    // 실체화된 Car 객체의 개수를 위한 정적 변수
    private static int numbers = 0;

    public Car(String m, String c, int s) {
        model = m;
        color = c;
        speed = s;
        // 자동차의 개수를 증가하고 id에 대입한다.
        id = ++numbers;
    }
        // 정적 메소드
    public static int getNumberOfCars() {
        return numbers; // OK!
    }
}
```

정적 변수 예제

```
public class CarTest {
    public static void main(String args[]) {
        Car c1 = new Car("S600", "white", 80);      // 첫 번째 생성자 호출
        Car c2 = new Car("E500", "blue", 20);       // 첫 번째 생성자 호출
        int n = Car.getNumberOfCars();              // 정적 메소드 호출
        System.out.println("지금까지 생성된 자동차 수 = " + n);
    }
}
```

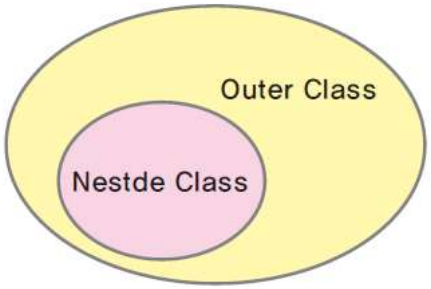

```
지금까지 생성된 자동차 수 = 2
```

내장 클래스

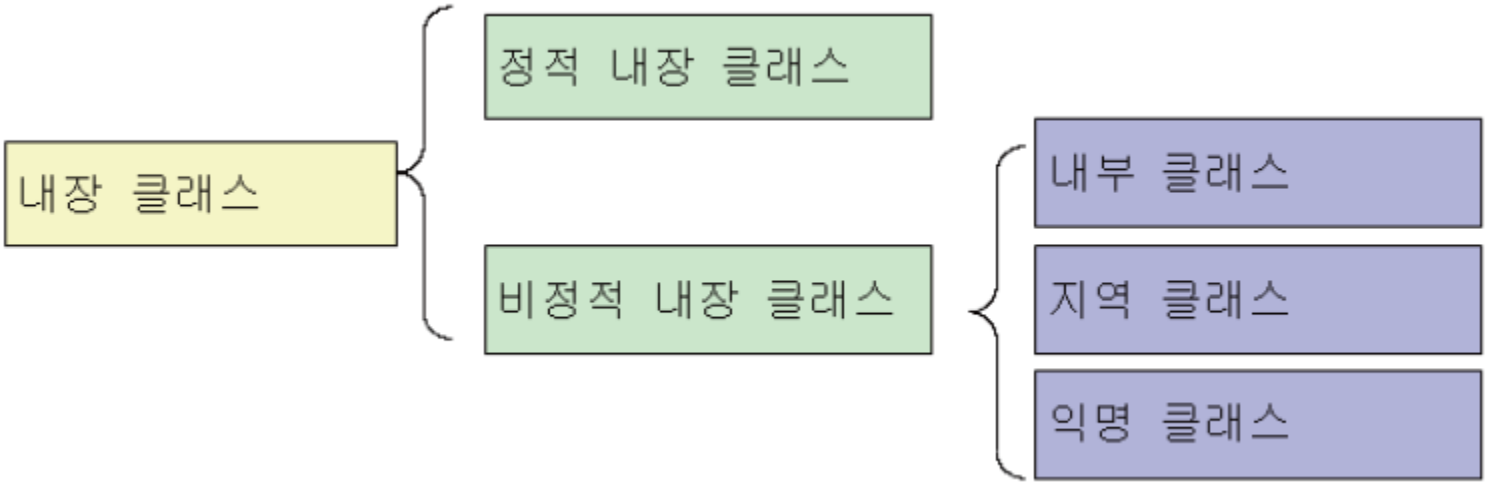
- 자바에서는 클래스 안에서 클래스를 정의할 수 있다.

형식

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```



내장 클래스의 분류



내부 클래스

- 클래스 안에 클래스를 선언하는 경우이다.

형식

```
class OuterClass {
    ...
    class InnerClass {  }
    ...
}
```

클래스가 다른 클래스 안에 내장된다.

정적 변수 예제

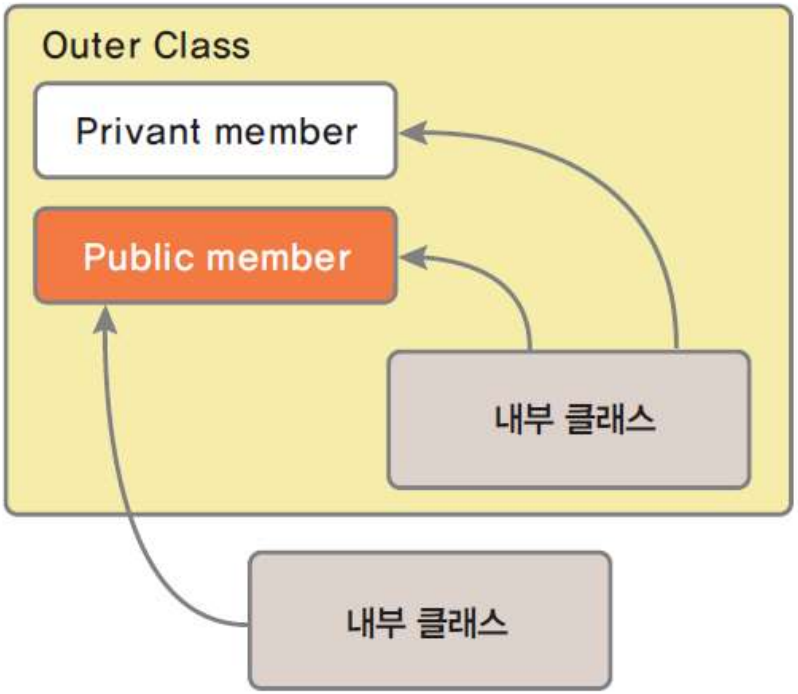
```
public class OuterClass {
        private int value = 10;

        class InnerClass {
                public void myMethod() {
                        System.out.println("외부 클래스의 private 변수 값:
" + value);
                }
        }

        OuterClass() {
                InnerClass obj = new InnerClass();
                obj.myMethod();
        }
}
```

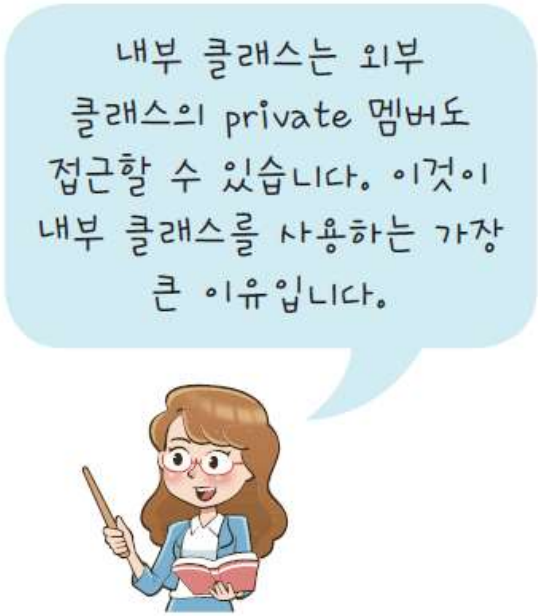
정적 변수 예제

```
public class InnerClassTest {
        public static void main(String[] args) {
                OuterClass outer = new OuterClass();
        }
}
```


```
외부 클래스의 private 변수 값: 10
```

내부 클래스를 사용하는 이유





Q & A