

자바 프로그래밍

제5장 클래스, 객체, 메소드

학습 내용

학습목차

- 01 객체 지향 프로그래밍이란?
- 02 객체 지향 프로그래밍의 특징
- 03 클래스 기초
 - LAB 객체 생성과 사용
- 04 메소드
 - LAB 자동차 클래스 작성하기
- 05 메소드 오버로딩
- 06 UML
- 07 String 클래스 사용
 - LAB String 클래스 활용

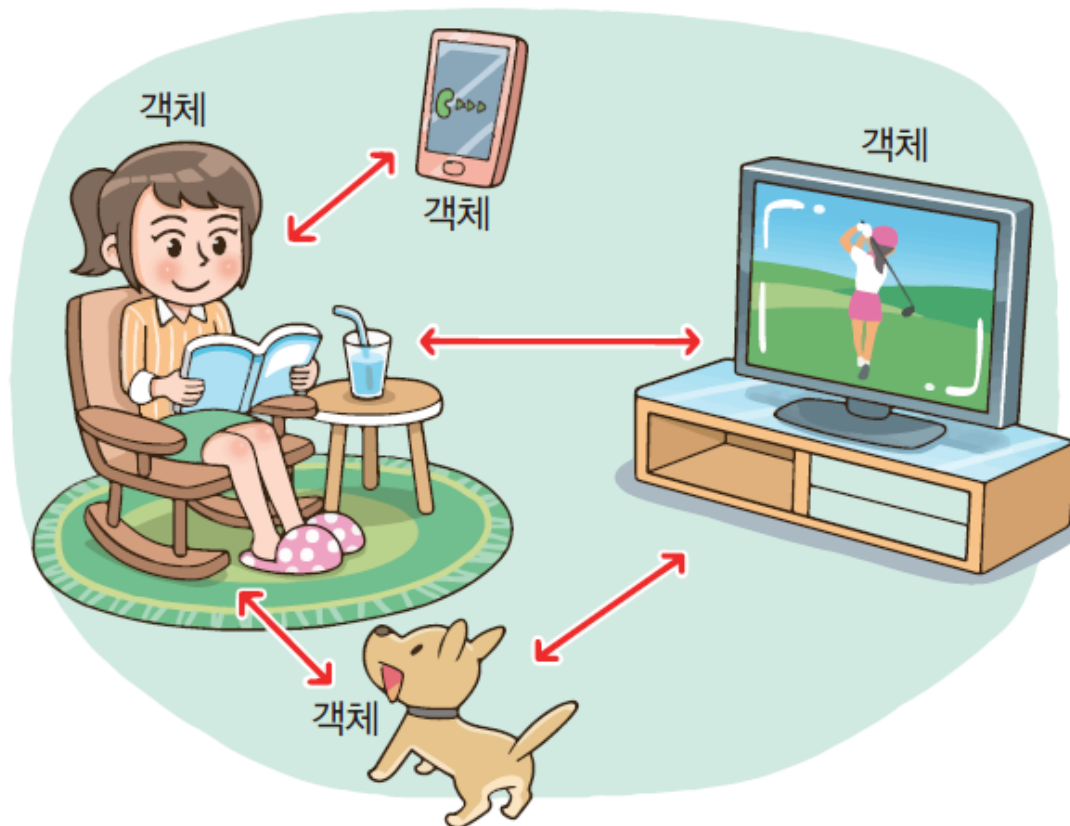
이번 장에서부터 드디어 객체
지향 프로그래밍이 시작되는 건
가요?

그렇습니다. 클래스, 객체, 메
소드는 자바 프로그래밍의 핵심입
니다. 이들 3가지에 대하여 확실히
이해하고 있어야 복잡한 프로
그램을 손쉽게 짤 수 있습니다.



객체

- 실제 세계는 **객체(object)**들로 이루어진다.



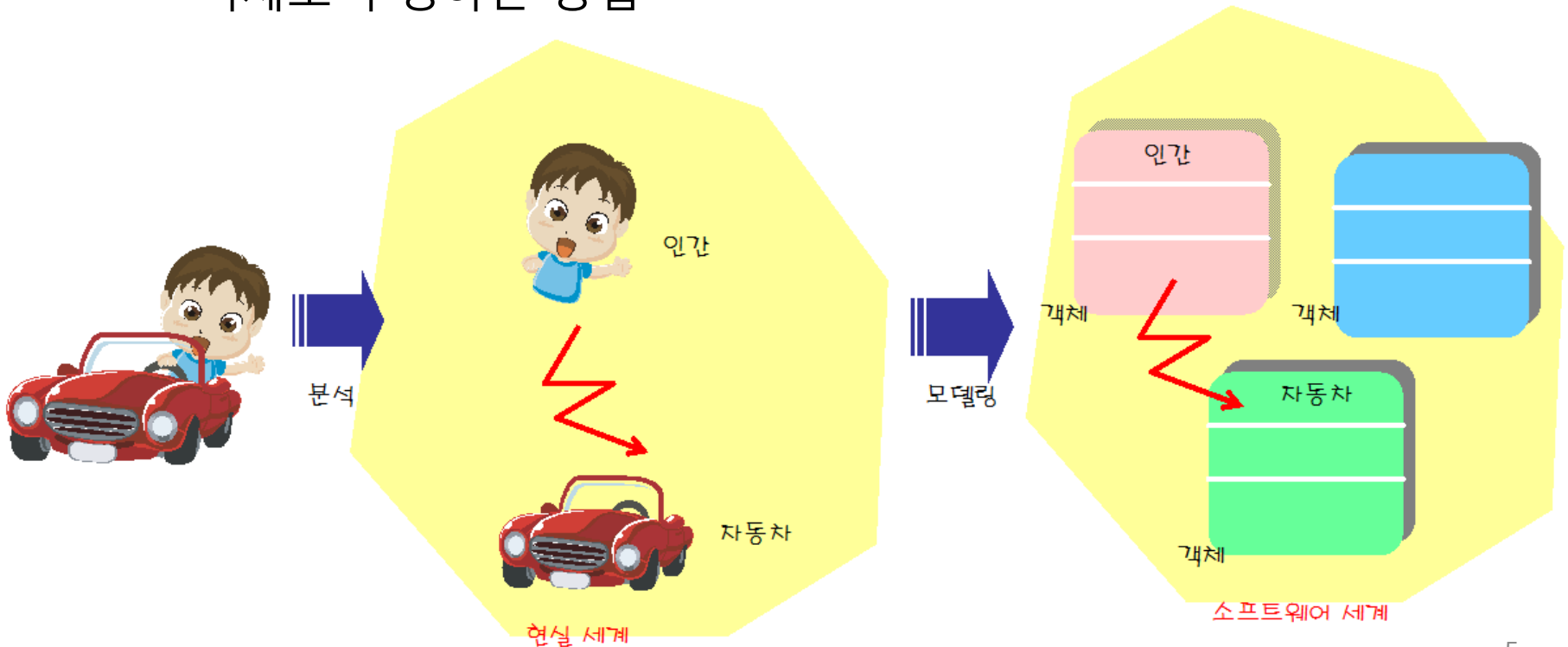
객체와 메시지

- 객체들은 메시지를 보내고 받으면서 상호작용한다.



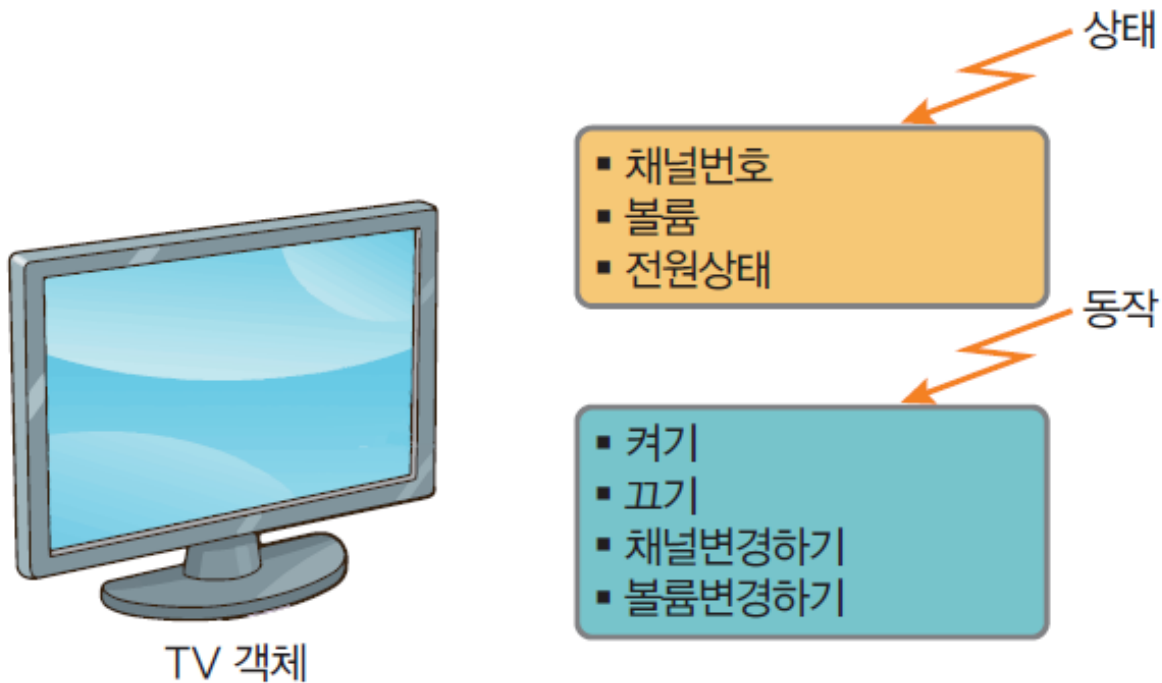
객체지향이란?

- 객체지향 프로그래밍(OOP: Object-Oriented Programming)
 - 실제 세계가 객체(object)로 구성된 것과 비슷하게, 소프트웨어도 객체로 구성하는 방법



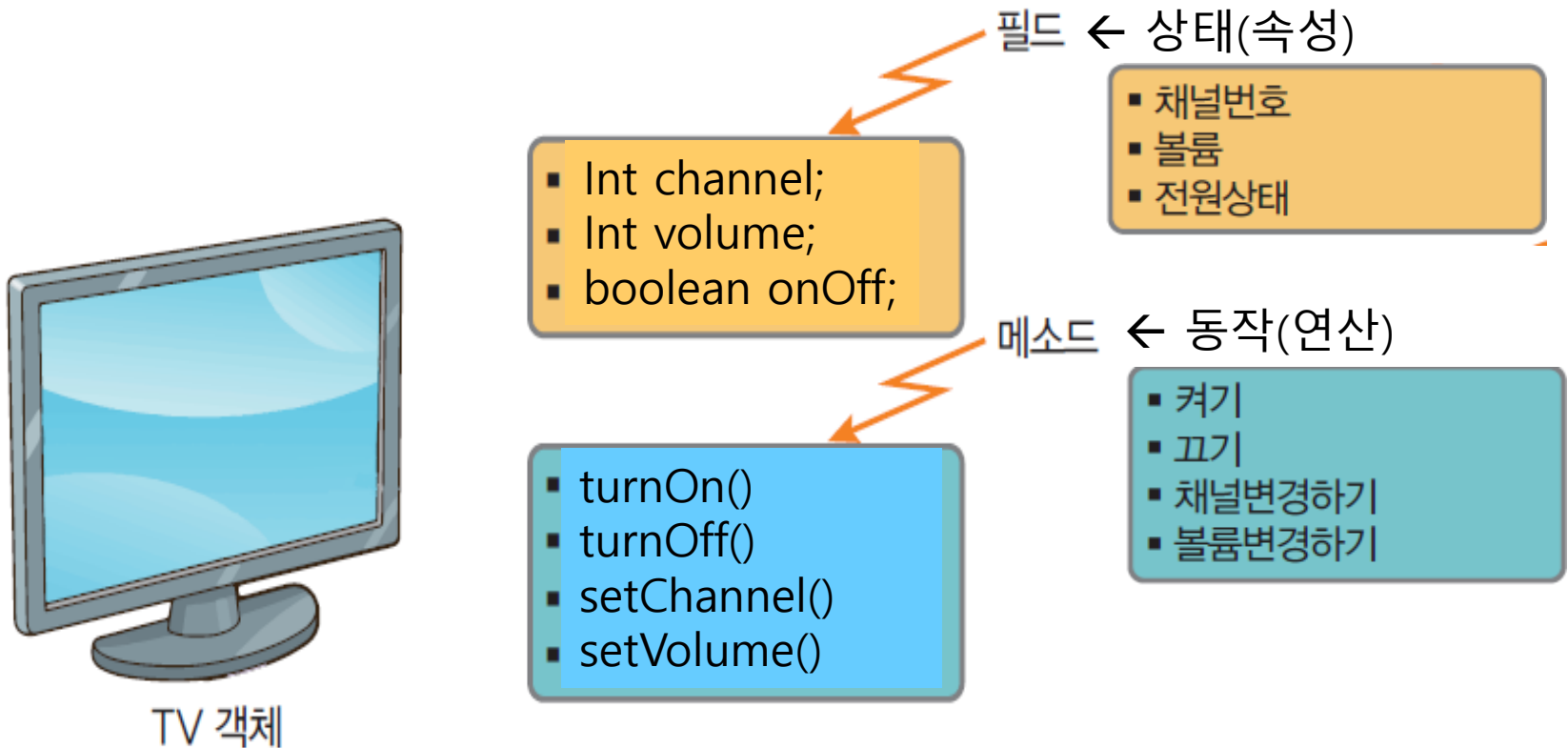
객체

- 객체는 상태(속성)와 동작(연산)을 가지고 있다.



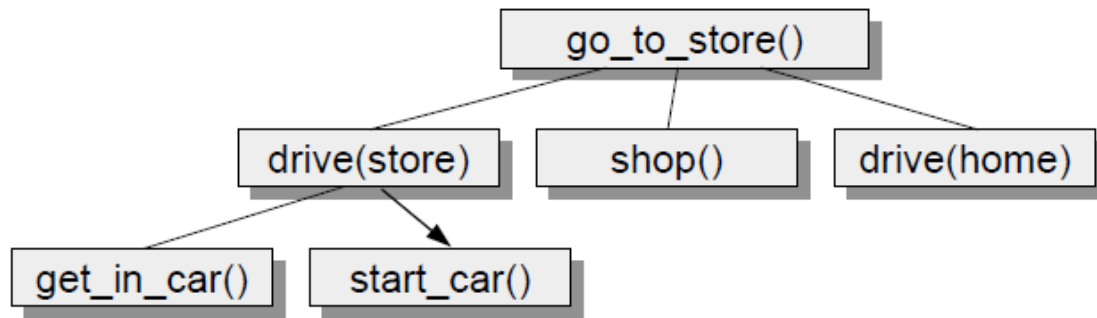
필드와 메소드

- 상태는 **필드(field)**로 구현하고, 동작은 **메소드(method)**로 구현한다.



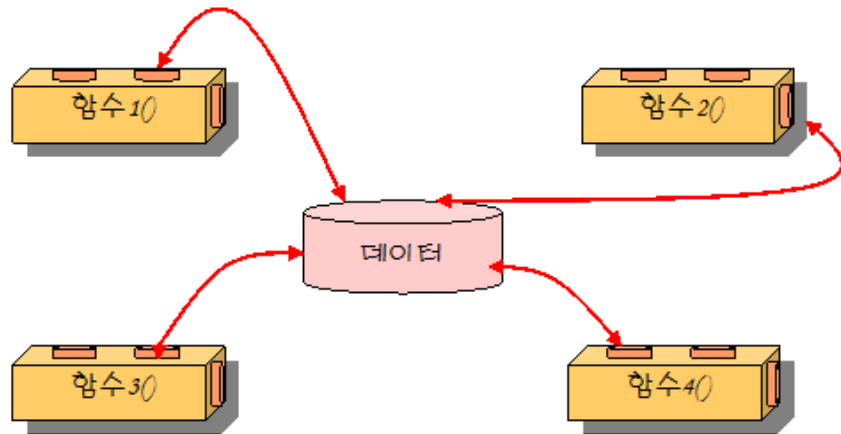
절차지향과 객체지향

- 절차지향 프로그래밍(procedural programming)
 - 문제를 해결하는 절차(프로시저)를 중심으로 생각하는 방법

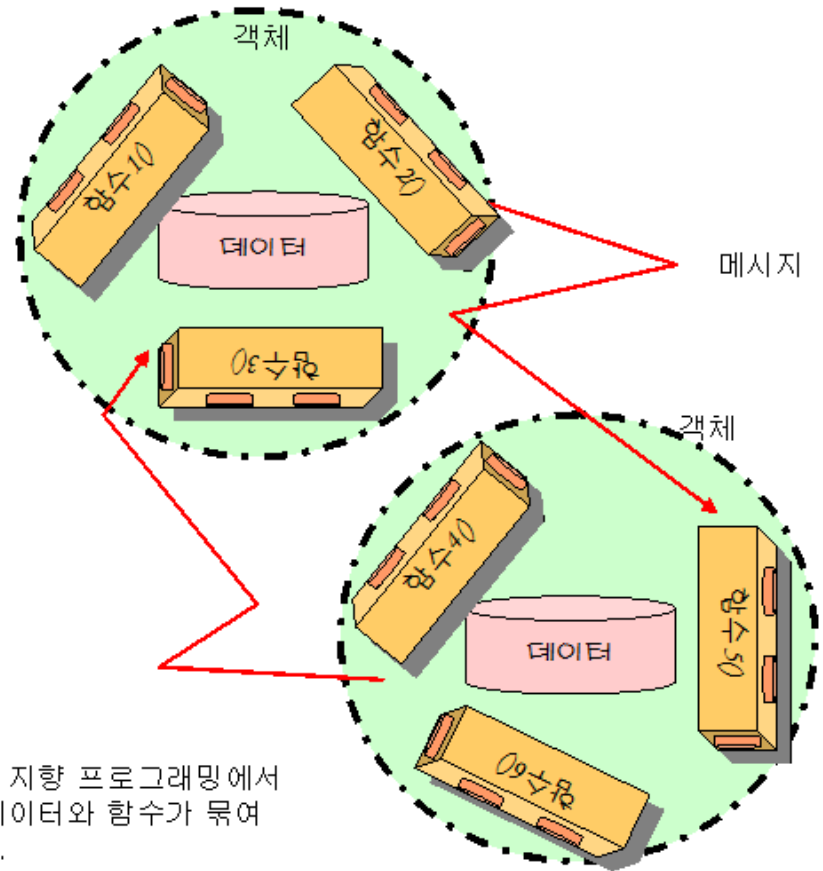


- 객체지향 프로그래밍(Object-Oriented Programming)
 - 데이터와 절차를 하나의 덩어리(객체)로 묶어서 객체를 중심으로 생각하는 방법

절차지향과 객체지향



절차 지향 프로그래밍에서
는 데이터와 함수가 묶여
있지 않다.



객체 지향 프로그래밍에서
는 데이터와 함수가 묶여
있다.

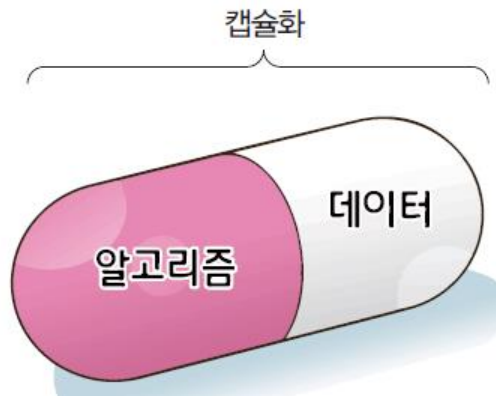
객체지향 프로그래밍의 특징

- 캡슐화(encapsulation)
- 상속(inheritance)
- 다형성(polymorphism)
- 추상화(abstraction)

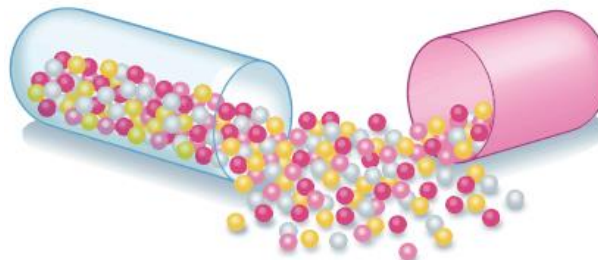
캡슐화

- 캡슐화(encapsulation)

- 관련된 데이터와 알고리즘(코드)이 하나의 묶음으로 정리되어 있는 것
- 객체가 하나의 캡슐



캡슐화는 데이터와 알고리즘을 하나로 묶는 것입니다.

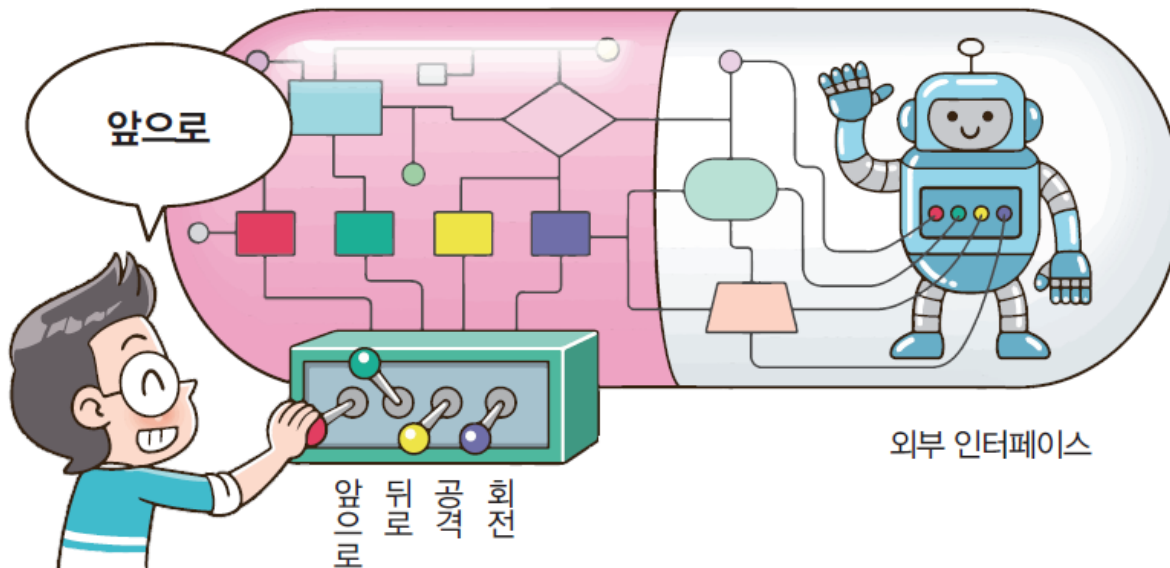


캡슐화 되어 있지 않은 데이터와 코드는 사용하기 어렵겠죠!



캡슐화와 정보은닉

- 정보 은닉(information hiding)
 - 객체를 캡슐화하여 객체의 내부를 보호하는 것
 - 즉, 객체의 실제 구현 내용을 외부에 감추는 것

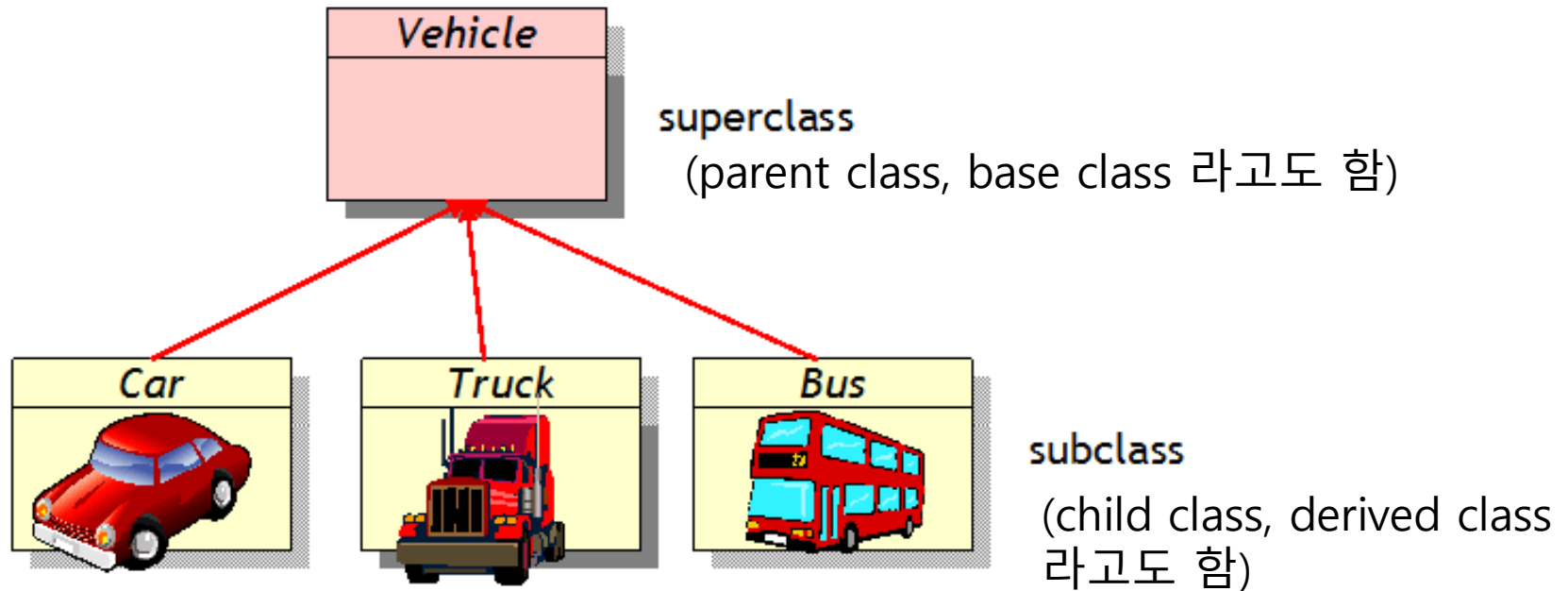


객체는 공개된
인터페이스를
통하여 사용
하여야 합니다.



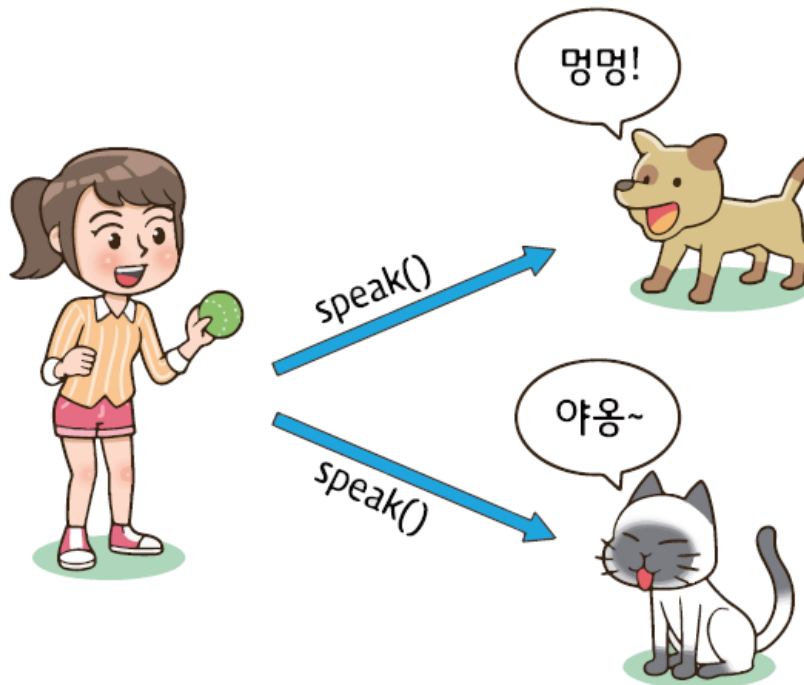
상속

- 상속(inheritance)
 - 이미 작성된 클래스(부모 클래스)를 이어받아서 새로운 클래스(자식 클래스)를 생성하는 기법



다형성

- 다형성(polymorphism)
 - 하나의 이름(방법)으로 많은 상황에 대처하는 기법



다형성은 객체의 동작이 상황에 따라서 달라지는 것을 말합니다. “speak”라는 메시지를 받은 객체들이 모두 다르게 소리를 내는 것이 바로 다형성입니다.



추상화

- 추상화(abstraction)
 - 불필요한 정보는 숨기고 중요한 정보만을 표현함으로써 프로그램을 간단히 만드는 기법



실제 객체



추상화된 객체

추상화는 필요한 것만을 남겨놓는 것입니다. 추상화 과정이 없다면 사소한 것도 신경 써야 합니다.



객체지향의 장점

- 신뢰성 있는 소프트웨어를 쉽게 작성할 수 있다.
- 코드를 재사용하기 쉽다.
- 업그레이드가 쉽다.
- 디버깅이 쉽다.

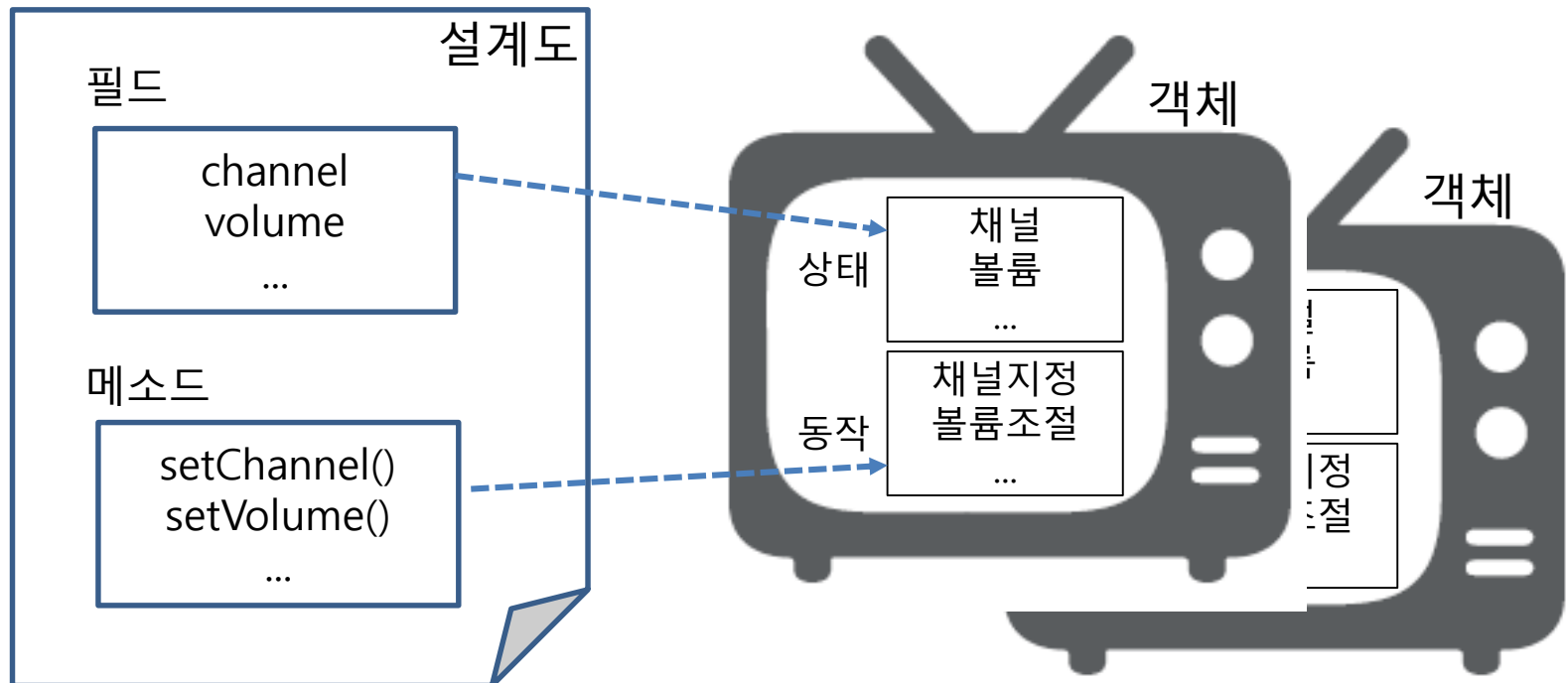
클래스와 객체

- **클래스(class)** : 객체를 만드는 설계도
- 클래스로부터 만들어지는 **객체(object)**를 그 클래스의 **인스턴스(instance)**라고도 한다.



클래스와 객체

- 클래스(설계도)는 그 클래스의 객체(인스턴스)가 지녀야 하는 상태와 동작을 정의한다.
 - 객체의 상태는 **필드(field)**로 정의
 - 객체의 동작은 **메소드(method)**로 정의



자바 클래스의 구조

```
class 클래스이름 {  
    // 필드 정의  
    type field1;  
    type field2;  
    ....  
  
    // 메소드 정의  
    type method1(...) { ... }  
    type method2(...) { ... }  
    ...  
}
```



```
class Television {  
    // 필드 정의  
    int channel;  
    int volume;  
    ...  
  
    // 메소드 정의  
    void setChannel(...) { ... }  
    void setVolume(...) { ... }  
    ...  
}
```

클래스 정의

- **필드(field)**는 클래스 안에 선언된 변수로서, 객체의 상태 (속성)을 나타냄
- 우선 필드만 갖는 텔레비전 클래스를 정의해보자.

클래스 {

Television.java

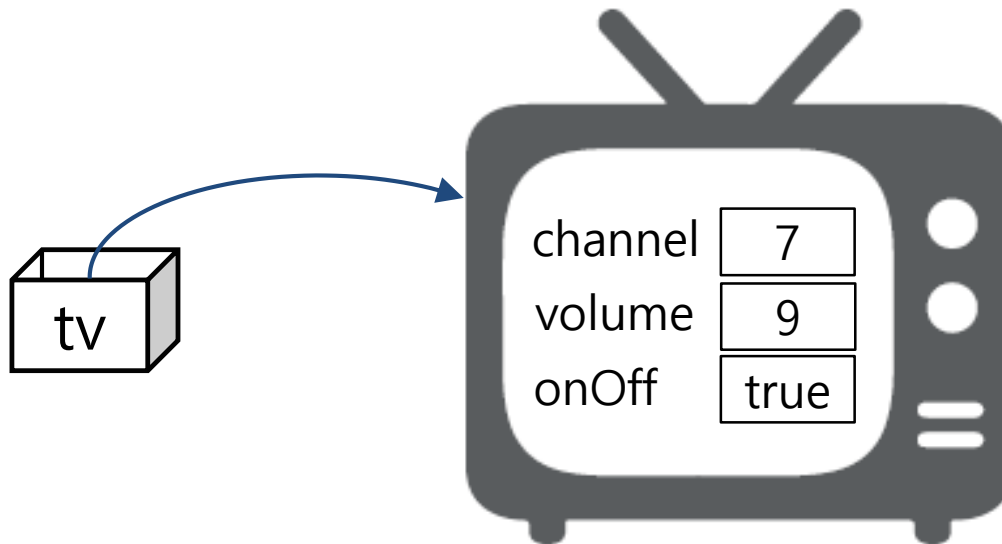
```
01 public class Television {  
02     int channel;        // 채널 번호  
03     int volume;         // 볼륨  
04     boolean onOff;      // 전원 상태  
05 }
```

필드 정의
객체의 속성을 나타낸다.

객체 생성

- **new** 연산자를 이용하여 객체 생성
- 텔레비전 객체를 생성하는 자바 코드

Television tv = **new** Television();



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.

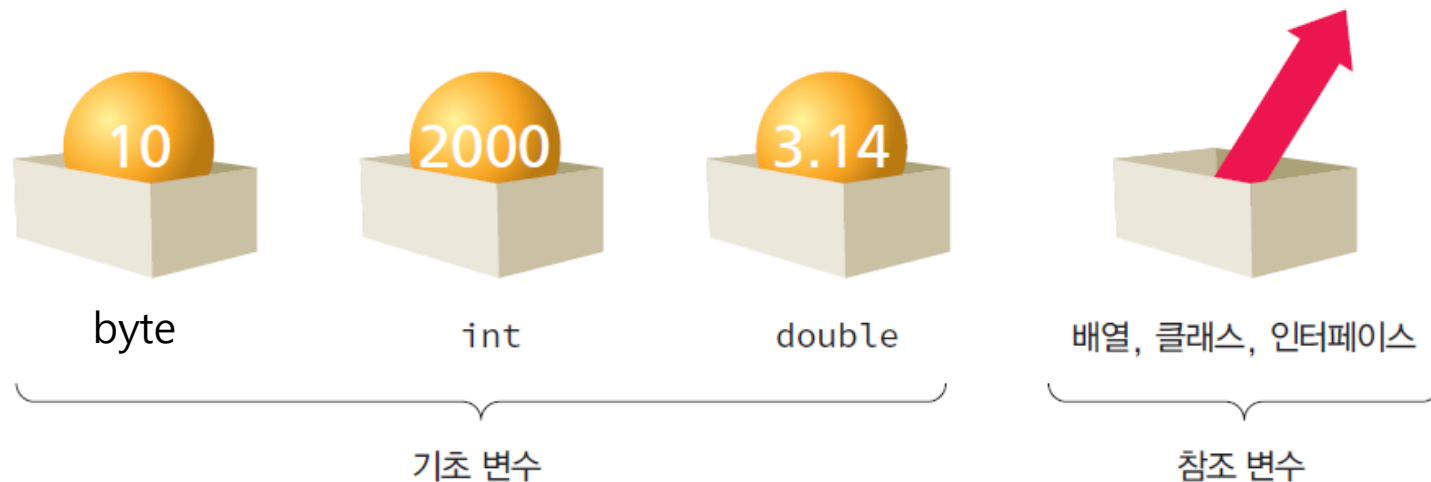
텔레비전의 채널은 7이고 볼륨은 9입니다.

Television.java

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

기초 변수와 참조 변수

- 기초 변수(primitive variable)
 - 실제 데이터 값이 저장된다.
- 참조 변수(reference variable)
 - 객체를 참조할 때 사용되는 변수로서 여기에는 객체의 참조값이 저장된다.

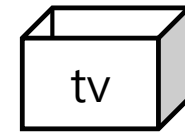


객체 생성 코드

Television tv = new Television();
로 표현할 수도 있다.

Television tv; // ① 참조 변수 선언
tv = new Television(); // ② 객체를 생성하여 ③ 참조값을 tv에 저장

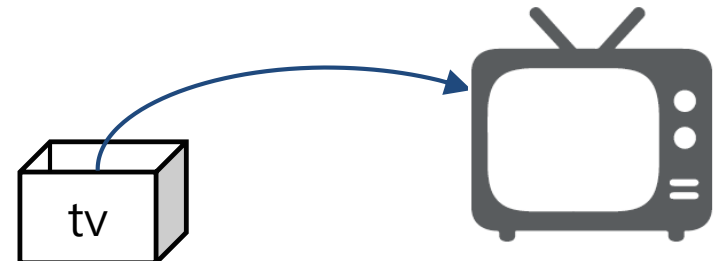
① 참조 변수 선언 – Television 타입의 객체를 참조할 수 있는 변수 tv를 선언한다.



② 객체 생성 – new 연산자를 이용하여 객체를 생성하고 객체 참조값(reference)을 얻는다.



③ 참조 변수와 객체의 연결 – 생성된 새로운 객체의 참조값을 tv 라는 참조 변수에 대입한다.



객체의 필드 사용

- 도트(.) 연산자를 사용하여 객체의 필드에 접근

```
Television tv = new Television();
```

```
tv.channel = 7;
```



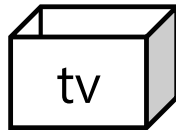
tv 객체의 channel 속성을
7로 지정

tv가 참조하는
객체의

channel 필드
에 접근

주의할 점

Television tv;



이 문장으로 객체가 생성되는 것은 아님.
Television 객체의 참조값을 담을 수 있는
변수만 생성됨

예) Television tv;
 tv.channel = 7; // 에러

예: 여러 개의 객체 생성

TelevisionTest.java

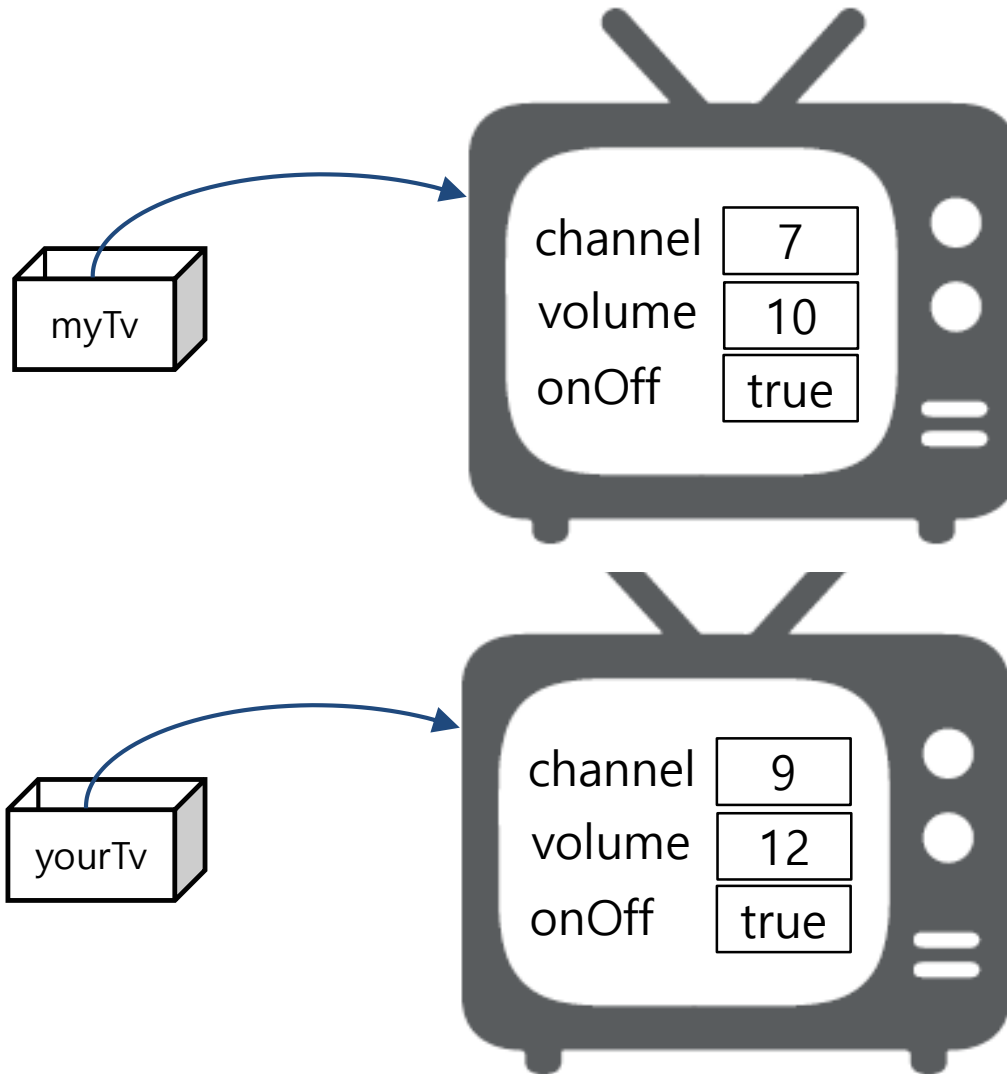
```
01 public class TelevisionTest {
02     public static void main(String[] args) {
03         Television myTv = new Television();
04         myTv.channel = 7;
05         myTv.volume = 10;
06         myTv.onOff = true;
07
08         Television yourTv = new Television();
09         yourTv.channel = 9;
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 }
```

객체

객체

myTv와 yourTv는 별개의 객체를 가리킨다.

객체 생성



각 객체마다 별도의
변수(필드)를 가짐

쓰레기 수집기

- 자바에서 객체는 new 연산자에 의해 힙 메모리(heap memory)에 할당됨
- 자동 쓰레기 수집(garbage collection)
 - C에서는 프로그래머가 직접 객체의 삭제를 책임져야 함
 - 자바에서는 자동 메모리 수거 시스템을 사용함
 - 참조가 전혀 없는 객체는 쓰레기로 간주하여 제거

```
Television myTv = new Television();
```

```
myTv = new Television();
```

```
...
```

```
myTv = null;
```

클래스와 소스 파일

- 일반적으로 하나의 소스 파일에 하나의 클래스만 넣는 것이 원칙이다.

```
// TelevisionTest.java
public class TelevisionTest {
}
```

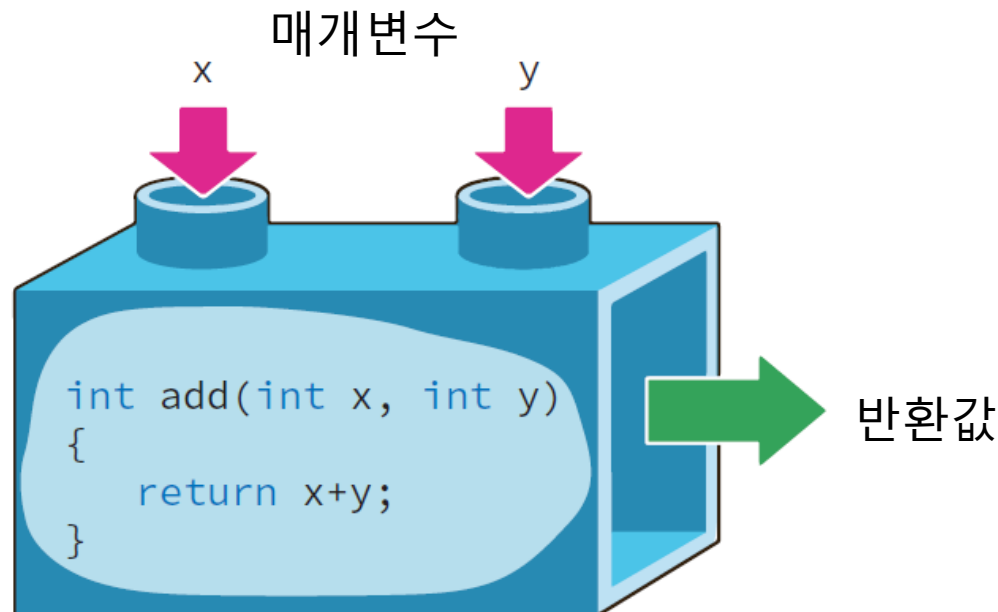
```
// Television.java
public class Television {
}
```

- 하나의 소스 파일에 여러 클래스를 넣을 수도 있다.
 - public 클래스의 이름과 소스 파일의 이름이 일치해야 함

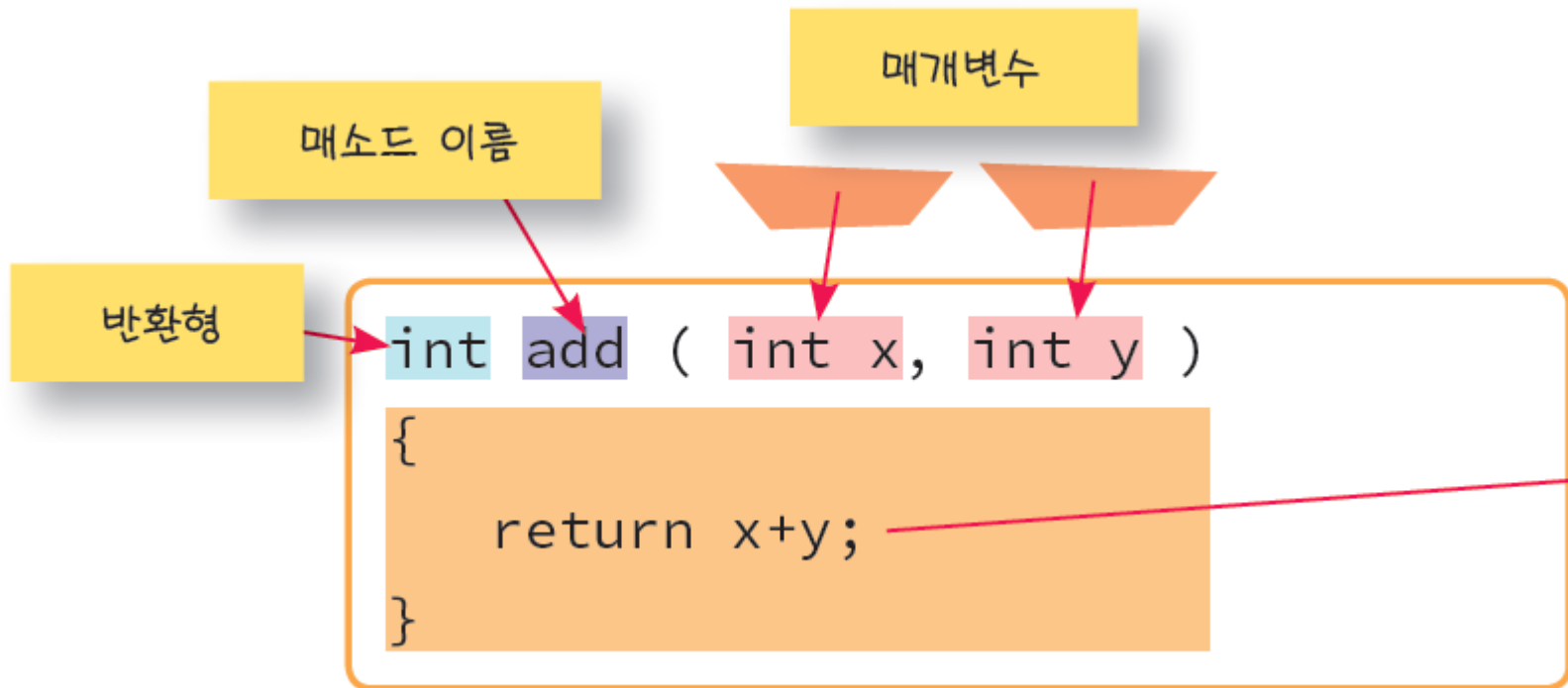
```
// TelevisionTest.java
public class TelevisionTest {
}
class Television {
}
```

메소드

- **메소드(method)**는 클래스 안에 정의된 특정한 작업을 수행하는 문장들의 모임으로서, 객체의 동작(연산)을 표현함
- 메소드는 매개변수를 받아서 처리를 하고 결과를 반환하는 상자



메소드의 구조



객체의 메소드 호출

- 도트(.) 연산자를 사용하여 객체의 메소드 호출

```
Television myTv = new Television();
```

myTv.print();

myTv가 참조하
는 객체의

print()를 호출



myTv 객체에게
print() 동작을 수행해
달라고 요청

예 : Television 클래스에 메소드 추가

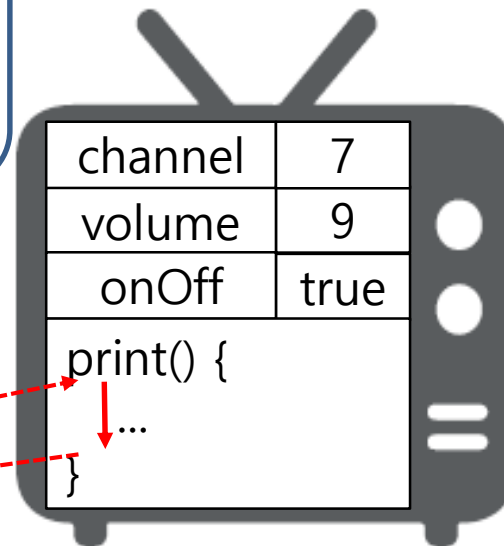
```
public class TelevisionTest {  
    public static void main(String[] args) {  
        Television myTv = new Television();  
        myTv.channel = 7;  
        myTv.volume = 9;  
        myTv.onOff = true;  
        myTv.print();  
    }  
}
```

```
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.");  
    }  
}
```

예제 설명

myTv.print() 문장이 실행되면
myTv 객체 안의 print()가 호출되
어 실행되고, 실행을 마치면
myTv.print() 문장으로 되돌아 옴

```
main() {  
  ...  
  myTv.print();  
  ...  
}
```

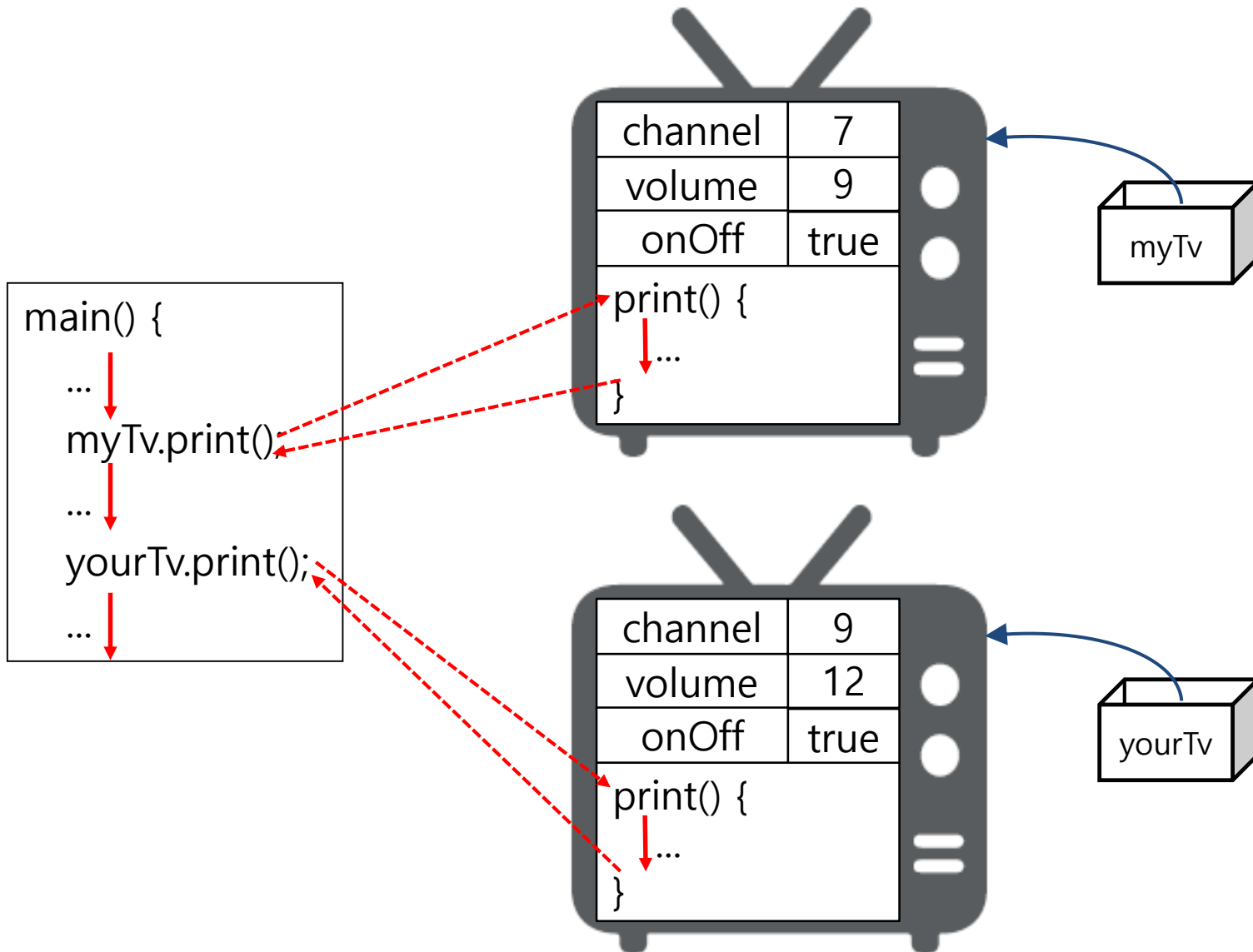


예 : 객체의 메소드 호출

```
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();  
    }  
}
```

```
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.");  
    }  
}
```

예제 설명



메소드의 반환

- return 문을 실행하면 메소드가 종료된다.

```
void myMethod() {  
    ...  
    return;  
    ...  
}
```

- return 문을 사용하여 메소드 종료와 동시에 반환값을 전달할 수 있다.

```
int yourMethod() {  
    ...  
    return a+b;  
    ...  
}
```

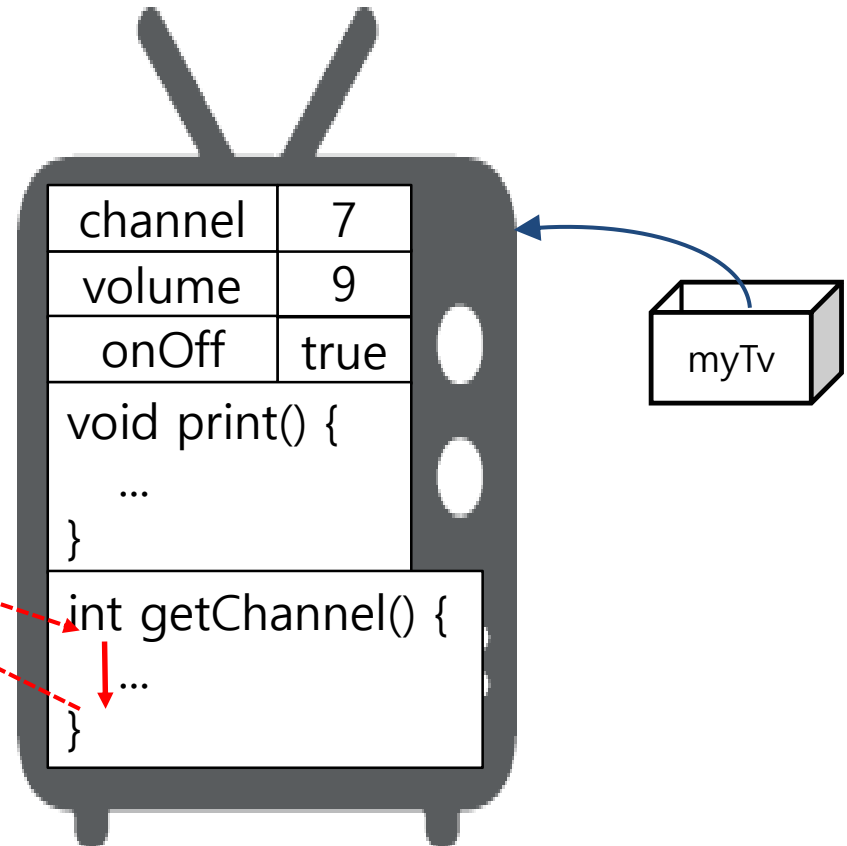
예: 반환값을 갖는 메소드

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

```
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;  
    }  
}
```

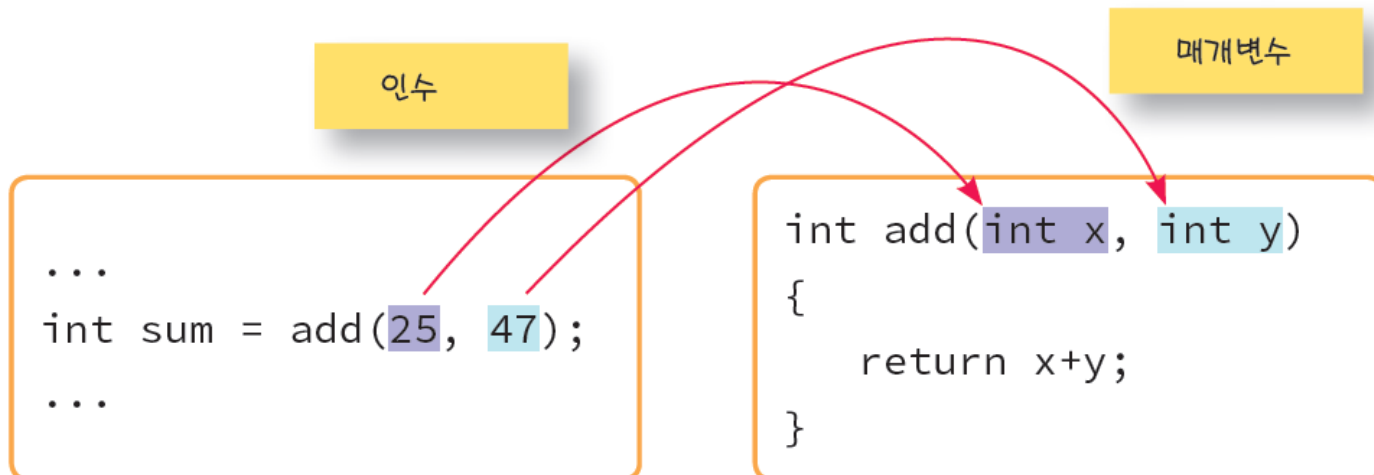
예제 설명

```
main() {  
    ...  
    ch = myTv.getChannel();  
    ...  
}
```



인수와 매개변수

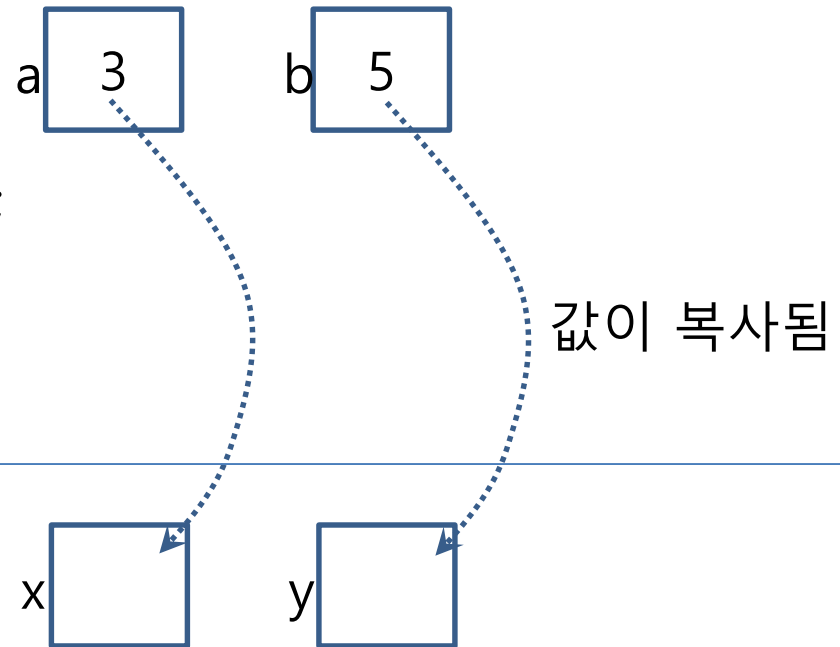
- 메소드 호출시 전달하는 값을 인수(argument)라고 함
 - 또는 실 매개변수(actual parameter)라고 함
- 호출시 전달된 값을 메소드에서 받을 때 사용하는 변수를 매개변수(parameter)라고 함
 - 또는 형식 매개변수(formal parameter)라고 함



값에 의한 전달(call-by-value)

```
void method1() {  
    int a = 3, b = 5;  
    int sum = add(a, b);  
    System.out.println(sum);  
    System.out.println(a);  
    System.out.println(b);  
}
```

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



매개변수 x와 y는 add() 내에서 일종의 지역변수(local variable)이다.

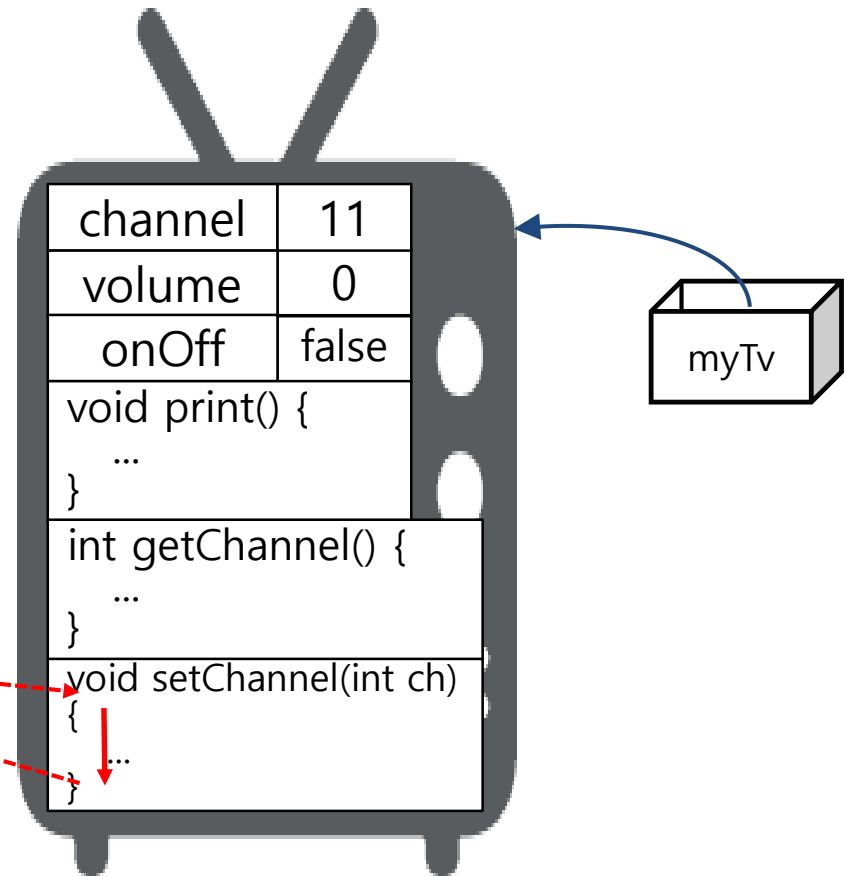
예 : 매개변수를 갖는 메소드

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

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

예제 설명

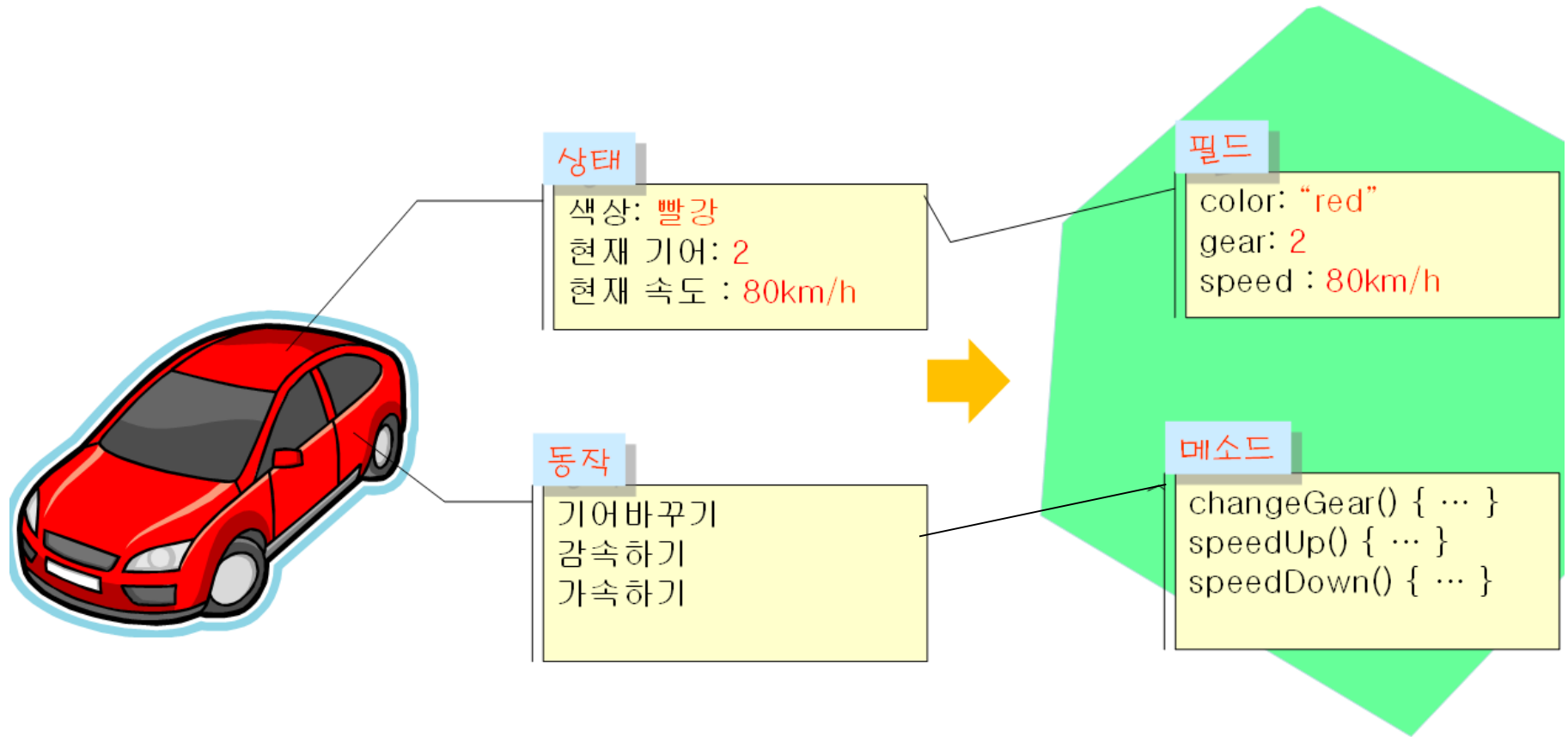
```
main() {  
  ...  
  myTv.setChannel(11);  
  ...  
}
```



```
public class TelevisionTest {  
    public static void main(String[] args) {  
        Television myTv = new Television();  
        Television yourTv = new Television();  
        myTv.setChannel(11);  
        yourTv.setChannel(9);  
        int ch1 = myTv.getChannel();  
        int ch2 = yourTv.getChannel();  
        System.out.println(ch1);  
        System.out.println(ch2);  
    }  
}
```

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

자동차 클래스 작성



예 : Car 클래스 작성과 객체 생성

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

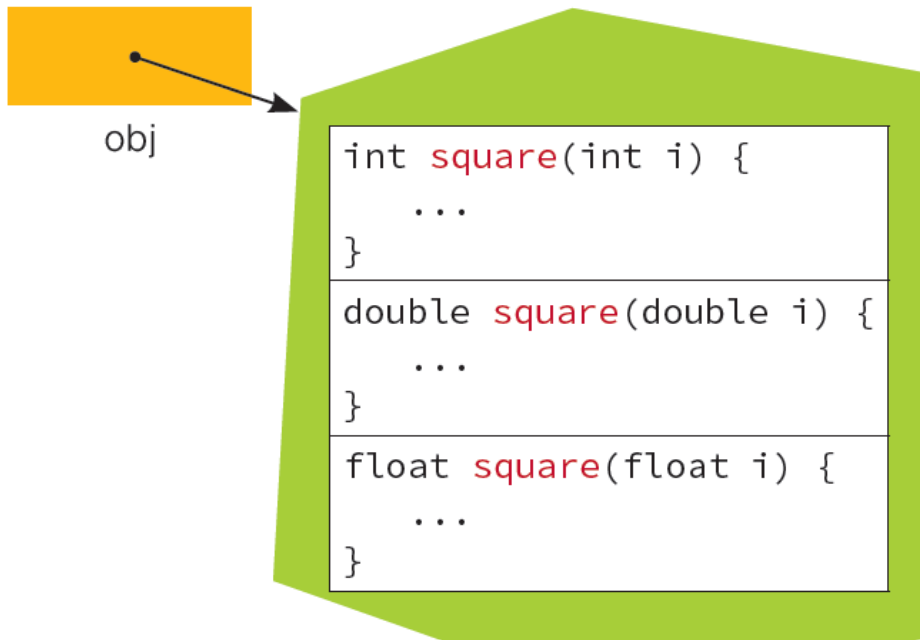
myCar.toString() 호출

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

```
public class Car {  
    String color; // 색상(초기값 null)  
    int speed;    // 속도(초기값 0)  
    int gear;     // 기어(초기값 0)  
  
    @Override  
    public String toString() {  
        return "Car [color=" + color +  
            ", speed=" + speed +  
            ", gear=" + gear + "];"  
    }  
  
    void changeGear(int g) {  
        gear = g;  
    }  
  
    void speedUp() {  
        speed = speed + 10;  
    }  
  
    void speedDown() {  
        speed = speed - 10;  
    }  
}
```

메소드 오버로딩

- **메소드 오버로딩(method overloading)**이란 하나의 클래스에 이름이 같은 메소드를 여러 개 두는 것
 - 각 메소드의 매개변수의 개수나 자료형이 달라야 함
 - 반환형(return type)만 다르게 하는 것은 인정되지 않음



메소드 오버로딩이란 이름이 같은 메소드를 여러 개 정의하는 것입니다. 다만 각각의 메소드가 가지고 있는 매개 변수는 달라야 합니다.

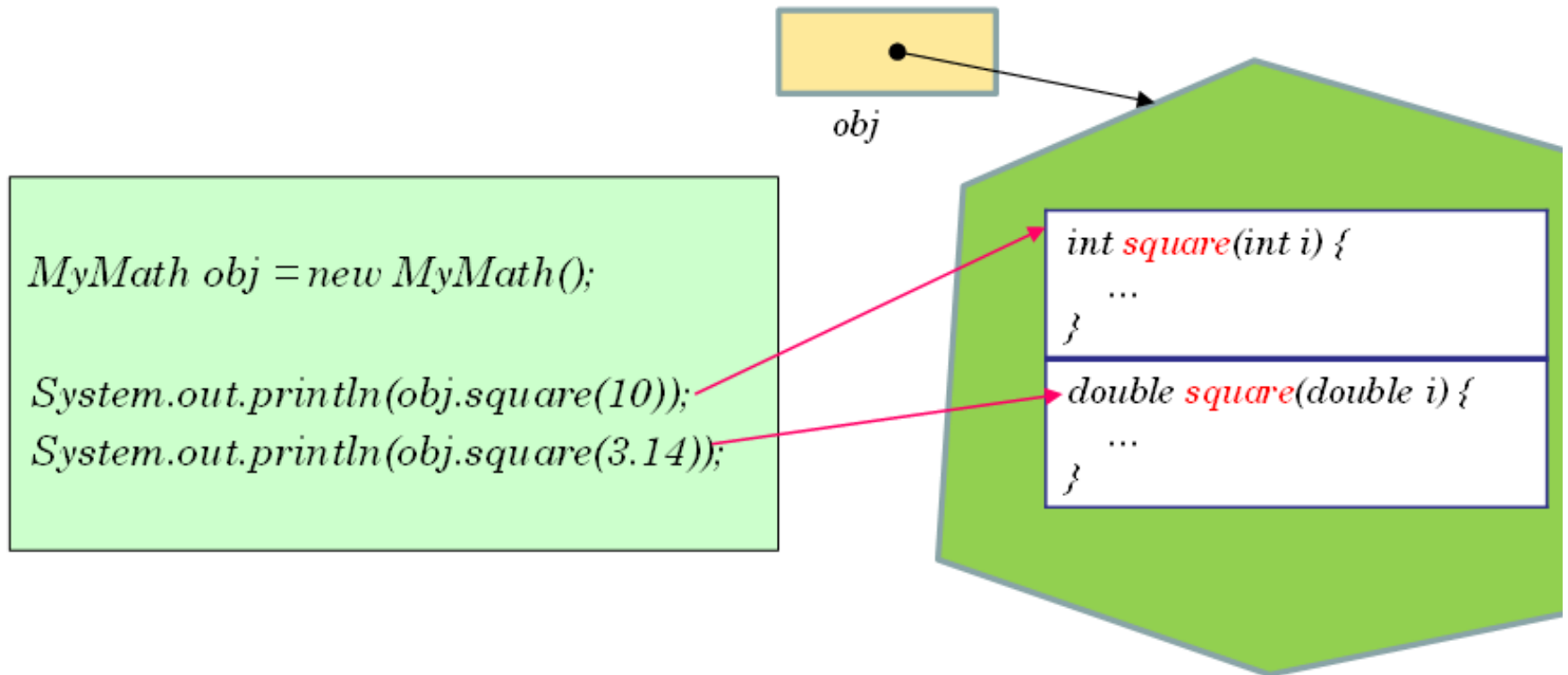


예 : 메소드 오버로딩

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

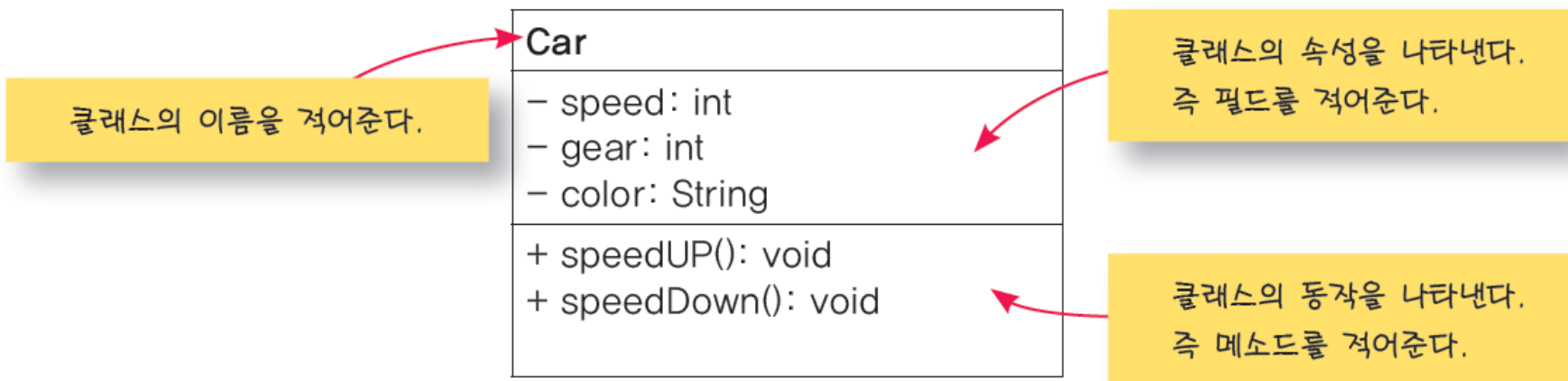
```
public class MyMathTest {  
    public static void main(String[] args) {  
        MyMath obj = new MyMath();  
        System.out.println(obj.square(10));  
        System.out.println(obj.square(3.14));  
    }  
}
```

예제 설명



UML

- UML(Unified Modeling Language)
 - 객체지향설계 시에 사용되는 모델링 언어
 - UML을 사용하면 소프트웨어를 본격적으로 작성하기 전에 구현하고자 하는 시스템을 시각화하여 검토할 수 있다.
- UML의 여러 다이어그램 중, 클래스 다이어그램에 대해 살펴본다.
- 예: Car 클래스의 클래스 다이어그램



UML : 가시성 표시자








- 필드나 메소드의 이름 앞에는 가시성 표시자(visibility indicator)가 올 수 있다.
 - 가시성 표시자는 6장에서 배움

+	Public
-	Private
#	Protected
/	Derived
~	Package

UML : 클래스 간의 관계

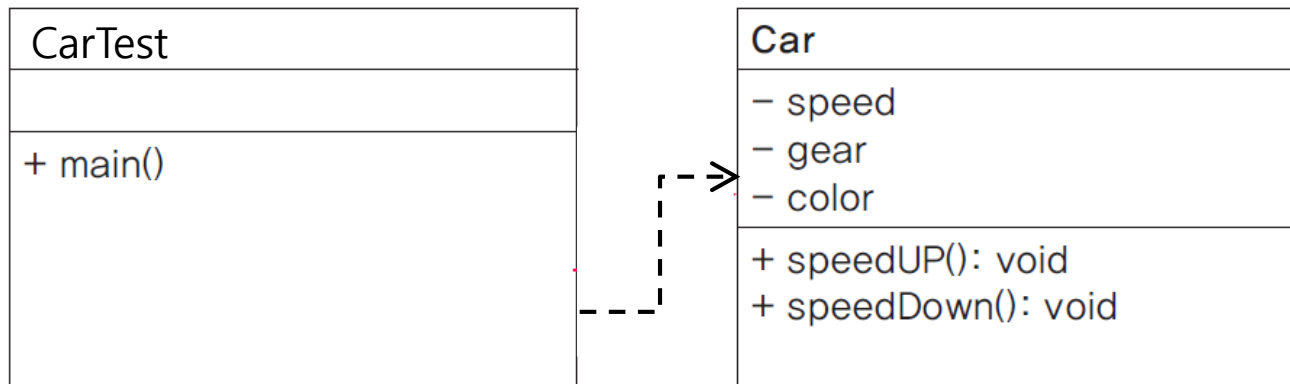
- 여러 종류의 화살표로 클래스 간의 관계를 나타낼 수 있다.

표 5-1 • UML에서 사용되는 화살표의 종류

관계	화살표
일반화(generalization), 상속(inheritance)	
구현(realization)	
구성관계(composition)	
집합관계(aggregation)	
유향 연관(direct association)	
양방향 연관(bidirectional association)	
의존(dependency)	

예 : 클래스 간의 관계

- 앞에서 다룬 Car 예제를 UML로 그려보면 다음과 같다.
 - 의존 관계(dependency) : CarTest 클래스가 Car 클래스를 사용

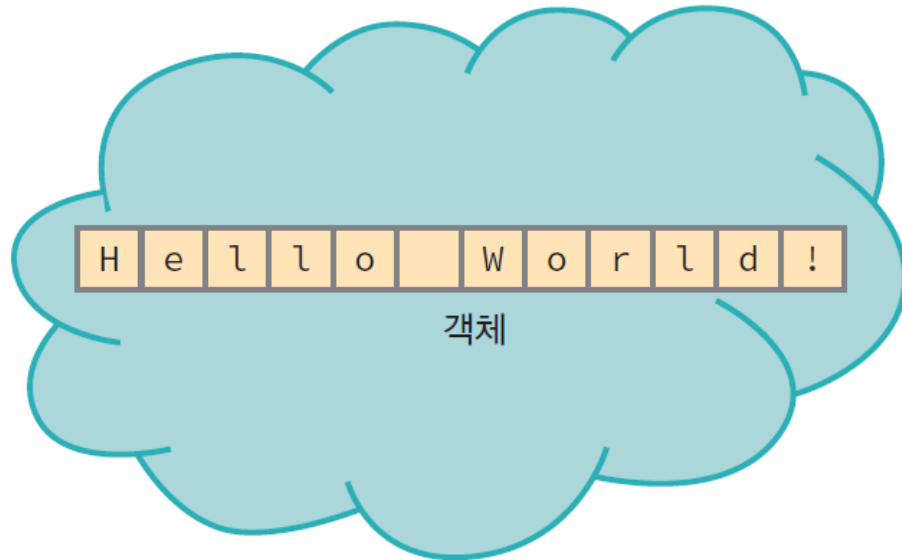


```
public class CarTest {  
    public static void main(String[] args) {  
        Car myCar = new Car();  
        myCar.speedup();  
        ...  
    }  
}
```

```
public class Car {  
    private speed;  
    ...  
    public void speedUp() {  
        speed = speed + 10;  
    }  
}
```

String 클래스

- 문자열은 자바에서 기초 자료형이 아니다.
- 대신 문자열을 저장하고 처리하는 String이라는 이름의 클래스가 존재한다.



heap 메모리

자바에서 문자열은
객체입니다.

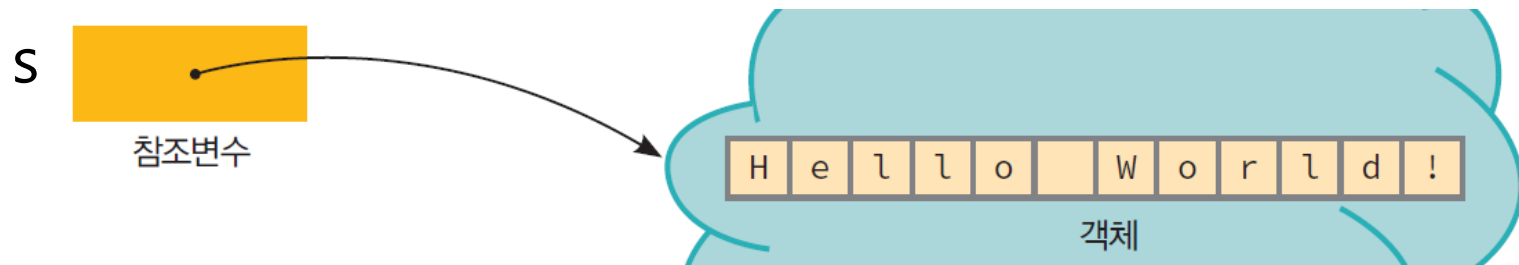


String 클래스의 객체 생성

- new 연산자를 이용하여 객체를 생성

Car c = **new** Car(); // Car 객체 생성

String s = **new** String("Hello World!"); // String 객체 생성



- 단, String 객체는 new 연산자를 사용하지 않고도 생성할 수 있다.

String s2 = "Hello World!";

문자열 상수로 표기해도 자동으로 객체가 생성됨

String 클래스의 메소드

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

String 클래스 사용

```
public class StringTest {  
    public static void main(String[] args) {  
        String proverb = "A barking dog"; // new 연산자 생략  
        String s1, s2, s3, s4;  
  
        s1 = proverb.concat(" never Bites!"); // 문자열 결합  
        s2 = proverb.replace('g', '*'); // 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);  
  
        System.out.println(proverb.length()); // 문자열의 길이  
        System.out.println(s3.equals("bar")); // 문자열 비교  
    }  
}
```

A barking dog never Bites!
A barkin* do*
bar
A BARKING DOG
13
true

문자열의 결합

- + 연산자를 사용하여 문자열을 결합할 수 있다.

```
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

- 문자열과 기초 자료형을 결합하면 자동으로 기초 자료형이 문자열로 변환된다.

```
int x = 20;  
int y = 30;  
String answer = "10" + x + y;  
System.out.println(answer);
```

102030

문자열 → 수치값

- 문자열 "123"을 숫자 123으로 변환하려면?
 - 랩퍼(wrapper) 클래스인 Integer 클래스를 이용한다.

```
String s1 = "123";  
int n1 = Integer.parseInt(s1); // n1에 정수 123이 저장됨
```

- 문자열 "3.14"를 숫자 3.14로 변환하려면?
 - 랩퍼(wrapper) 클래스인 Double 클래스를 이용한다.

```
String s2 = "3.14";  
double n2 = Double.parseDouble(s2); // n2에 실수 3.14가 저장됨
```

객체이름.메소드이름() 형식이 아니라
클래스이름.메소드이름() 형식의 메소드 호출이다.
이에 대해서는 뒤에서 배움

래퍼 클래스

- 기초 자료형에 해당하는 래퍼 클래스(wrapper class)

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

래퍼 클래스는 기초 자료형을 클래스로 만들고 싶은 경우에 사용하면 됩니다. 문자열을 수치값으로 변환해주는 메소드도 가지고 있습니다.



래퍼 클래스

- Integer 클래스 사용 예

```
Integer obj = new Integer(10);
```

- Integer 클래스의 메소드

반환값	메소드 이름	설명
byte	<code>byteValue()</code>	int형을 byte형으로 변환
double	<code>doubleValue()</code>	int형을 double형으로 변환
float	<code>floatValue()</code>	int형을 float형으로 변환
int	<code>parseInt(String s)</code>	문자열을 int형으로 변환
String	<code>toBinaryString(int i)</code>	int형의 정수를 2진수 형태의 문자열로 변환
String	<code>toHexString(int i)</code>	int형의 정수를 16진수 형태의 문자열로 변환
String	<code>toOctalString(int i)</code>	int형의 정수를 8진수 형태의 문자열로 변환

LAB: String 클래스 활용

- 반복하여 사용자에게 문자열을 받아서 문자열이 "www."로 시작하는지를 검사하는 프로그램을 작성해 보자.
 - 사용자가 "quit"를 입력하면 프로그램을 종료한다.

```
문자열을 입력하세요> www.google.com
www.google.com 은 'www.'로 시작합니다.
문자열을 입력하세요> naver.com
naver.com 은 'www.'로 시작하지 않습니다.
문자열을 입력하세요> quit
```



```

import java.util.Scanner;

public class StringTest {
    public static void main(String[] args) {
        String str;
        Scanner input = new Scanner(System.in);

        while (true) {
            System.out.print("문자열을 입력하세요> ");
            str = input.next();
            if (str.equals("quit") == true)
                break;
            if (str.matches("^www\\.\\.(.+)")) {
                System.out.println(str + " 은 'www.'로 시작합니다.");
            } else {
                System.out.println(str + " 은 'www.'로 시작하지 않습니다.");
            }
        }
    }
}

```