

COMP217

Java Programming

Spring 2018

Week 9

*Fields and Methods, Constructors, Access
Control*

Goals

- 이번주에 배우게 될 내용 :
 - 클래스와 객체
 - 메소드와 필드
 - UML
 - 생성자
 - 정적 변수와 정적 메소드
 - 접근제어
 - this
 - 클래스간의 관계
- 파워 자바 8장(필드와 메서드)
- 파워 자바 9장(생성자와 접근제어)

Fields and Methods

Power Java 2, Chapter 8

변수의 종류

```
class
{
    public int speed
    ...
    void start(int s)
    {
        int t;
        ...
    }
}
```

field: 클래스 안에서 선언되는
멤버 변수, 인스턴스
변수라고도 한다.

필드

parameter: 메소드
선언에서의 변수

매개변수

지역 변수

local variable:
메소드나 블록 안에서
선언되는 변수

필드의 선언



필드의 접근 지정자는 어떤 클래스가 필드에 접근할 수 있는지를 표시한다.

- ▶▶ **public:** 이 필드는 모든 클래스로부터 접근가능하다.
- ▶▶ **private:** 클래스 내부에서만 접근이 가능하다.

필드의 사용 범위

Date.java

```
01 public class Date {
02
03     public void printDate() {
04         System.out.println(year + "." + month + "." + day);
05     }
06
07     public int getDay() {
08         return day;
09     }
10
11     // 필드 선언
12     private int year;
13     private String month;
14     private int day;
15 }
```


선언 위치와는 상관없이 어디서나
사용이 가능하다.

필드의 초기화

- 선언과 동시에 초기화 가능

```
public class Classroom {  
    public static int capacity = 60;        // 60으로 초기화  
    private boolean use = false;           // false로 초기화  
}
```

- 선언만 하고 초기화를 하지 않는다면 default값으로 초기화
- BUT!! 지역 변수는 사용하기 전에 반드시 명시적으로 초기화를 시켜줘야 함
- 즉, 아래 코드는 컴파일 오류 발생

```
public static void main(String args []) {  
    int n;   
    System.out.println(n);  
}
```

설정자와 접근자(Setter and getter)

- 설정자(mutator)
 - 보통은 setter라고 불림
 - 필드의 값을 설정하는 메소드
 - setXXX() 형식
- 접근자(accessor)
 - 보통 getter라고 불림
 - 필드의 값을 반환하는 메소드
 - getXXX() 형식



그림 8-1 . 접근자와 변경자 메소드만을 통하여 필드에 접근한다.


```

01 class Car {
02     private String color;    // 색상
03     private int speed;      // 속도
04     private int gear;       // 기어
05     public String getColor() {
06         return color;
07     }
08     public void setColor(String c) {
09         color = c;
10     }
11     public int getSpeed()    { return speed; }
12     public void setSpeed(int s) { speed = s; }
13     public int getGear()    { return gear; }
14     public void setGear(int g) { gear = g; }
15 }
16 public class CarTest1 {
17     public static void main(String[] args) {
18         // 객체 생성
19         Car myCar = new Car();
20
21         myCar.setColor("red");
22         myCar.setSpeed(100);
23         myCar.setGear(1);
24
25         System.out.println("현재 자동차의 색상은 " + myCar.getColor());
26         System.out.println("현재 자동차의 속도는 " + myCar.getSpeed());
27         System.out.println("현재 자동차의 기어는 " + myCar.getGear());
28     }
29 }

```

필드가 모두 private로 선언되었다.
클래스 내부에서만 사용이 가능하다.

color에 대한 접근자 메소드

color에 대한 설정자 메소드


이런 식으로 간략하게
표기하기도 한다.

객체 생성

설정자 메소드 호출

setter getter 예제

setter와 getter는 왜 사용하는가?

- 접근이 불가능한 필드에 대한 접근 방법 제공
- 매개 변수를 통하여 잘못된 값이 넘어오는 경우, 이를 사전에 차단할 수 있다.
- getter만을 제공하면 자동적으로 읽기만 가능한 필드를 만들 수 있다. 
- 필요할 때마다 필드값을 계산하여 반환할 수 있다.

```
public void setSpeed(int s)
{
    if( s < 0 )
        speed = 0;
    else
        speed = s;
}
```

속도가 음수이면 0으로 만든다.

지역 변수

- 메소드 안에 선언
- 메소드의 매개 변수도 지역 변수의 일종

```
class Box {  
    int width=0, length=0, height=0;  
    ...  
    public int getVolume()  
    {  
        int volume;  
        volume = width*length*height;  
        return volume;  
    }  
}
```

필드: 전체 클래스 안에서 사용가능

지역 변수

지역 변수 volume의 사용 범위

주의

- 지역 변수를 초기화하지 않고 사용하면 오류

```
class BugClass {  
    public int getSum() {  
        int sum;  
        for (int i = 0; i < 10; i++)  
            sum += i;  
        return sum;  
    }  
}
```

초기화되지 않은 지역변수를 사용하면 오류 발생

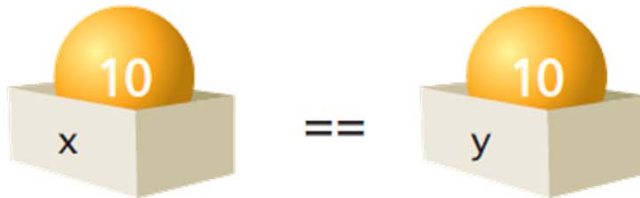
실행결과

Exception in thread "main" java.lang.Error: Unresolved compilation problems:
The local variable sum may not have been initialized

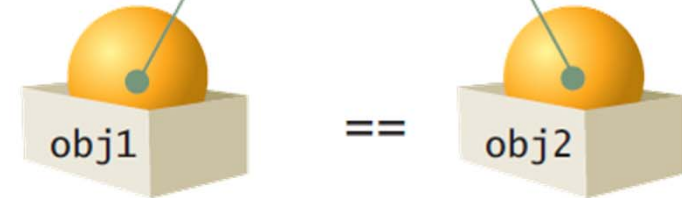
변수와 변수의 비교



- "변수1 == 변수2"의 의미



기초형 변수의 경우
값이 같으면 true



참조형 변수의 경우
같은 객체를 가리키면 true

- 참조형 변수의 경우, 객체의 내용이 같다는 의미가 아니다.

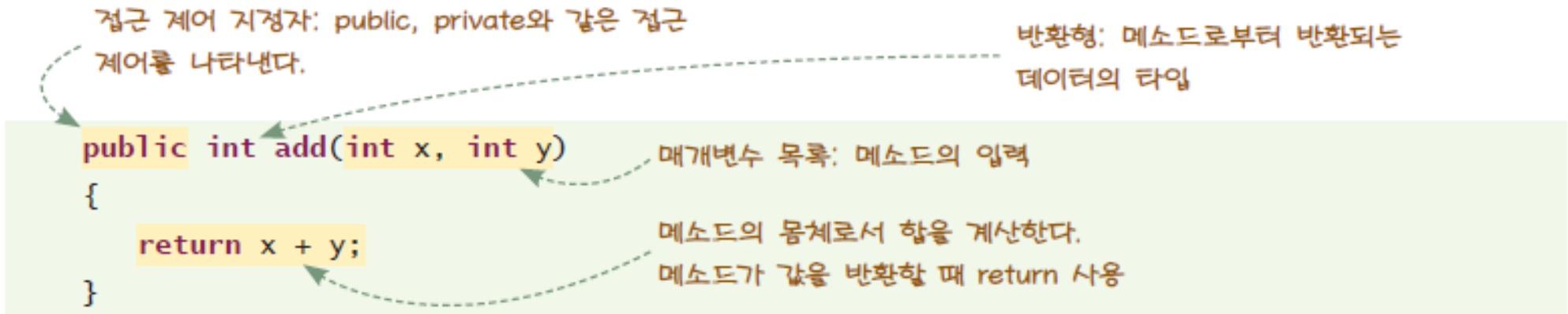
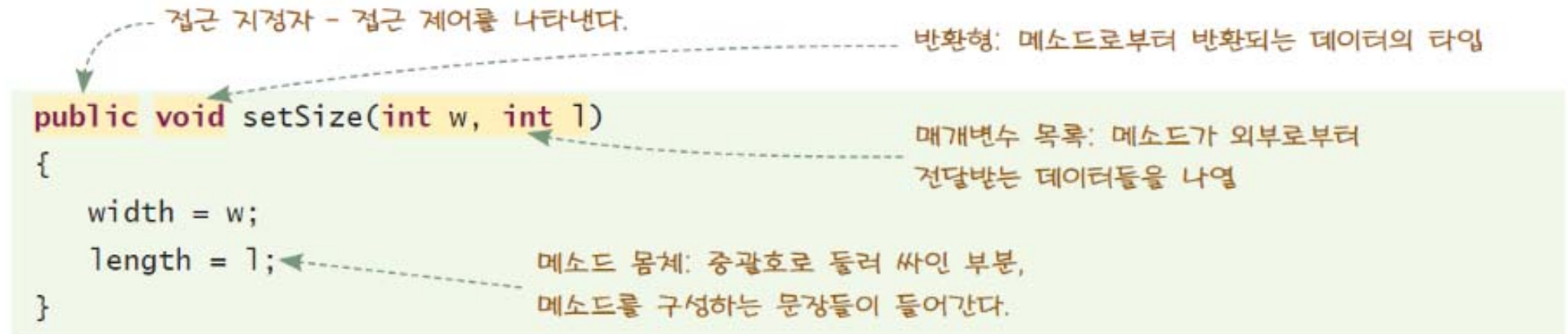
- 같은 메모리를 가리켜야 한다.

```
String a = s.nextLine();  
if(a=="abc") ...;  
else ...;
```

- 내용이 같은지를 검사하려면 equals() 사용

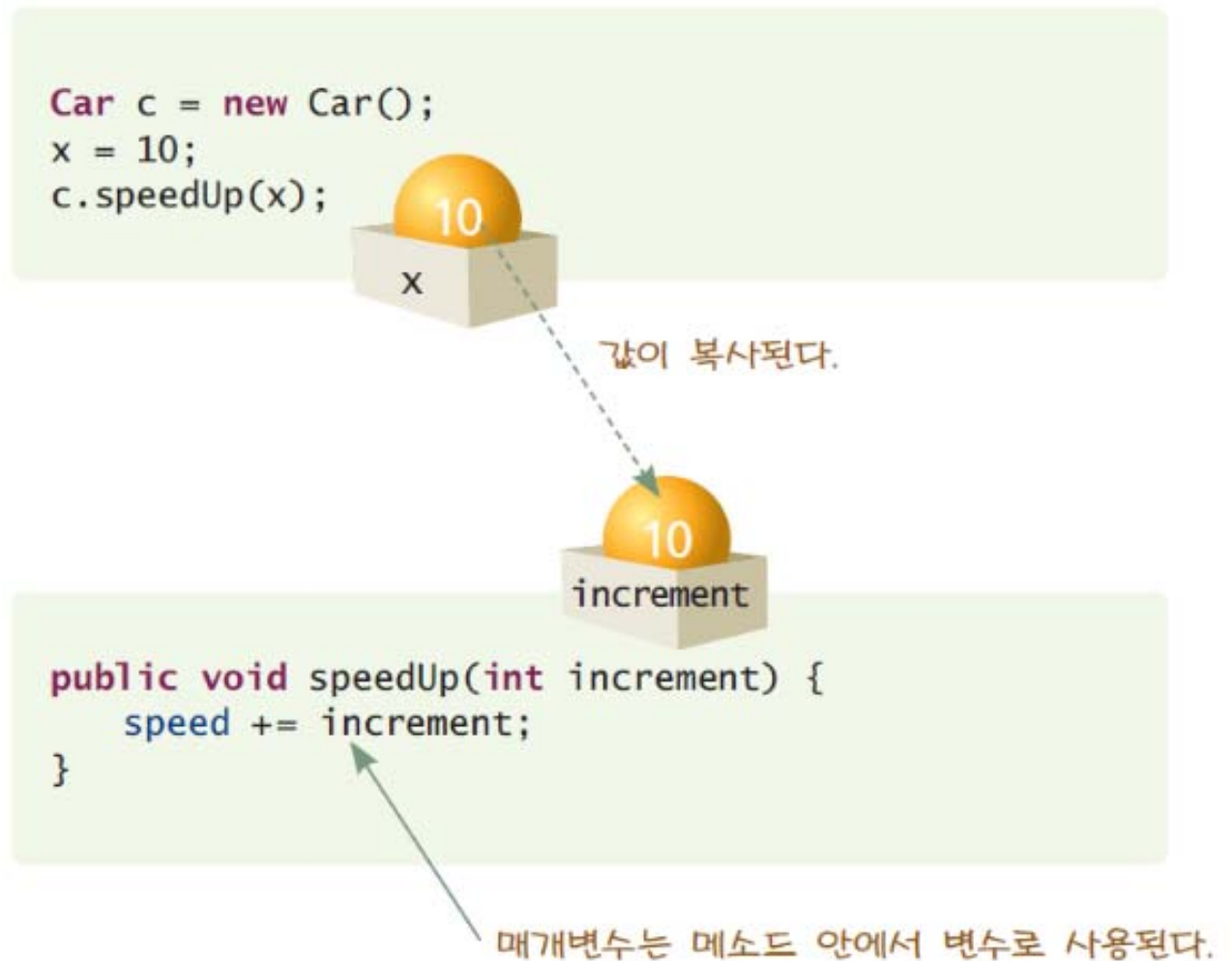
메소드

- 메소드는 객체가 할 수 있는 기능을 나타낸다.
- 클래스 안에 포함된 함수-> 메소드



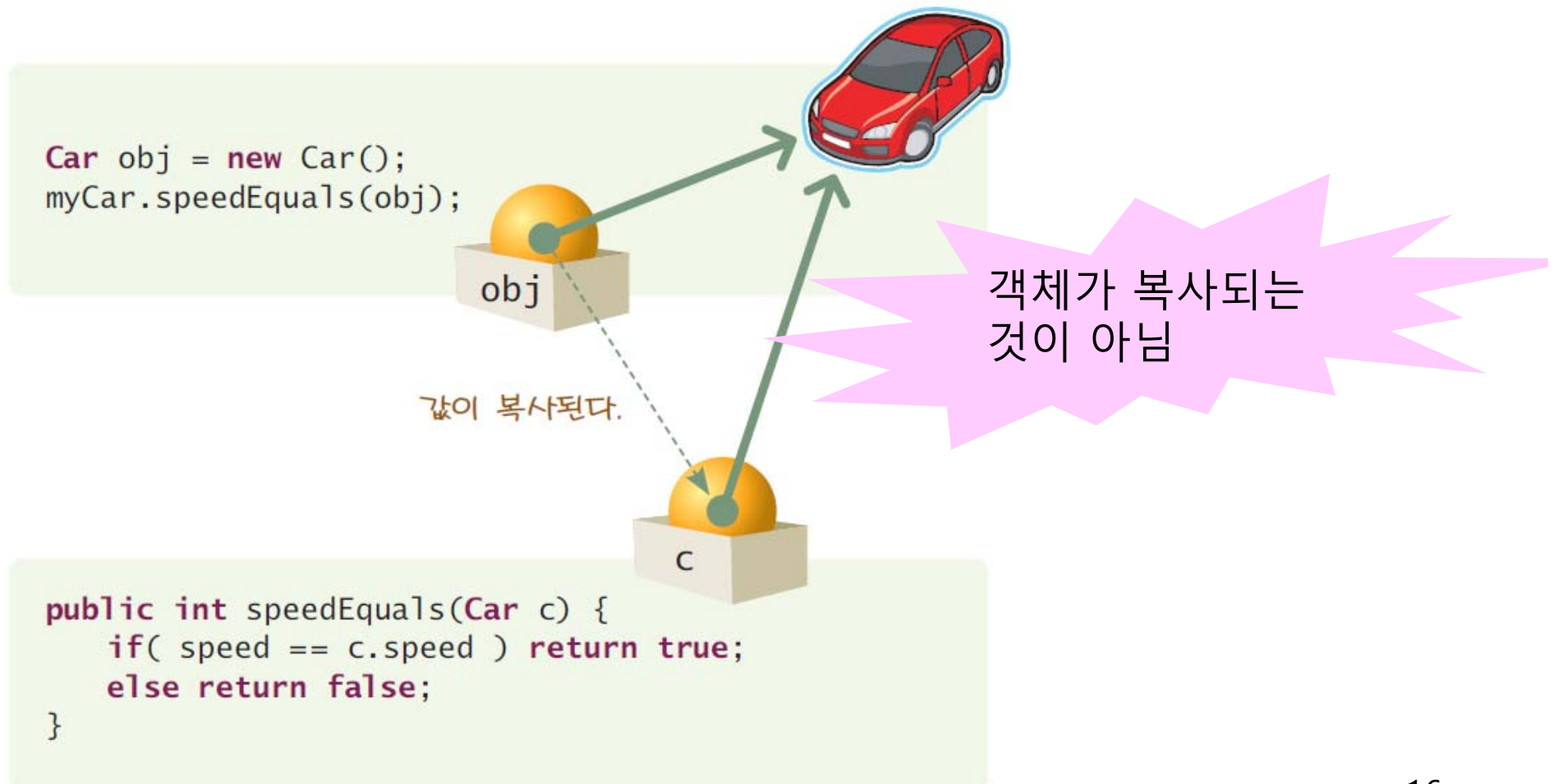
값에 의한 전달

- 매개 변수가 기초형의 변수일 경우, 값이 복사된다.



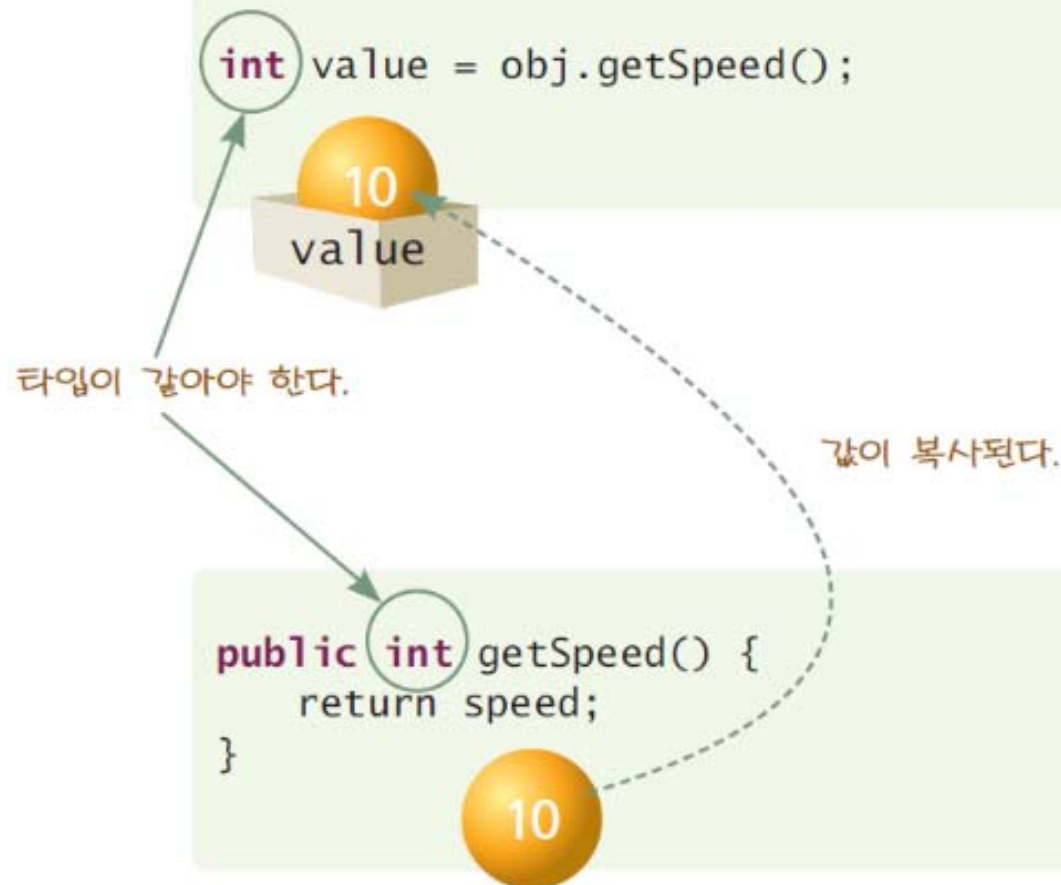
매개 변수가 객체인 경우

- 매개 변수가 참조형의 변수일 경우에는 참조값이 복사된다.

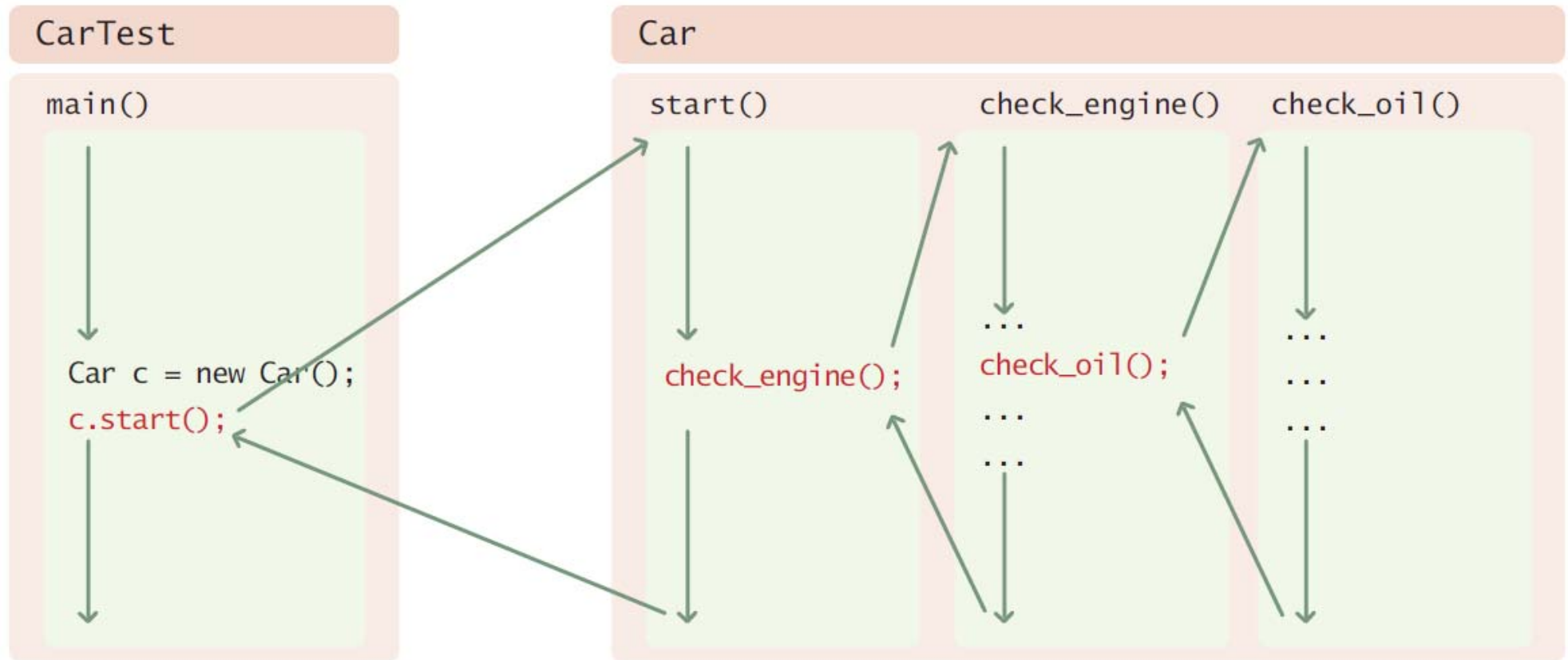


메소드는 값을 반환할 수 있다.

- 메소드는 작업의 결과값을 반환할 수 있다.
- 반환값은 하나만 가능하다.



메소드 호출



메소드가 다른 클래스에
있으면 객체를 통하여 호출

메소드가 같은 클래스에 있으면
메소드 이름을 통하여 호출

메소드 호출 예제

0.0부터 1.0 사이의
난수 값을 발생한다.



```

01 import java.util.*;
02 class DiceGame {
03
04     int diceFace;
05     int userGuess;
06
07     private void RollDice()
08     {
09         diceFace = (int)(Math.random() * 6) + 1;
10     }
11     private int getUserInput(String prompt)
12     {
13         System.out.println(prompt);
14         Scanner s = new Scanner(System.in);
15         return s.nextInt();
16     }
17     private void checkUserGuess()
18     {
19         if( diceFace == userGuess )
20             System.out.println("맞았습니다");
21         else
22             System.out.println("틀렸습니다");
23     }
24     public void startPlaying()
25     {
26         userGuess = getUserInput("예상값을 입력하시오: ");
27         RollDice();
28         checkUserGuess();
29     }
30 }

```

```

32 public class DiceGameTest {
33     public static void main(String[] args) {
34         DiceGame game = new DiceGame();
35         game.startPlaying();
36     }
37 }

```

DiceGame 객체를 생성한다.

다른 클래스에 있는 메소드는
객체를 적어주어야 한다.

같은 클래스에 있는 메소드는
객체를 적어줄 필요가 없다.

실행결과

예상값을 입력하시오:

3

틀렸습니다

중복 메소드(overloaded method)

- 메소드 호출시 매개 변수를 보고 일치하는 메소드가 호출된다.

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

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

매개변수만 다르면 메소드 이름은 같아도 된다.
이것을 중복 메소드라고 한다.

- 만약 square(3.14)와 같이 호출되면 컴파일러는 매개 변수의 개수, 타입, 순서 등을 봐서 두 번째 메소드를 호출한다.

중복 메소드 예제

CarTest2.java

```
01 class Car {
02     // 필드 선언
03     private int speed;        // 속도
04     // 중복 메소드: 정수 버전
05     public void setSpeed(int s) {
06         speed = s;
07         System.out.println("정수 버전 호출");
08     }
09
10     // 중복 메소드: 실수 버전
11     public void setSpeed(double s) {
12         speed = (int)s;
13         System.out.println("실수 버전 호출");
14     }
15 }
16
17 public class CarTest2 {
18     public static void main(String[] args) {
19         Car myCar = new Car();        // 첫 번째 객체 생성
20         myCar.setSpeed(100);           // 정수 버전 메소드 호출
21         myCar.setSpeed(79.2);         // 실수 버전 메소드 호출
22     }
23 }
```

실행결과

정수 버전 호출
실수 버전 호출



Method Overloading

```
1 public class TestMethodOverloading {
2     public static void main(String[] args) {
3         System.out.println("max(3, 4) = " + max(3, 4));
4         System.out.println("max(3.0, 5.4) = " + max(3.0, 5.4));
5         System.out.println("max(3.0, 5.4, 10.14) = "
6             + max(3.0, 5.4, 10.14));
7     }
8
9
10    public static int max(int num1, int num2) {    // (1)
11        if (num1 > num2) return num1;
12        else return num2;
13    }
14
15
16    public static double max(double num1, double num2) {    // (2)
17        if (num1 > num2) return num1;
18        else return num2;
19    }
20
21
22    public static double max(
23        double num1, double num2, double num3) {    // (3)
24        return max(max(num1, num2), num3);
25    }
26 }
27
```

가변 길이 인수

JDK 5부터 가변 길이 인수(variable-length arguments) 사용 가능

```
void sub(int a){  
    System.out.println(a);  
}
```

```
void sub(int a, int b){  
    System.out.println(a+" "+b);  
}
```

```
void sub(int a, int b, int c){  
    System.out.println(a+" "+b+" "+c);  
}
```

...??

15 }

16 }

" + v.length);

가변 길이 인수로 몇 개의
인수라도 받을 수 있다.

실행결과

인수의 개수 : 1

1

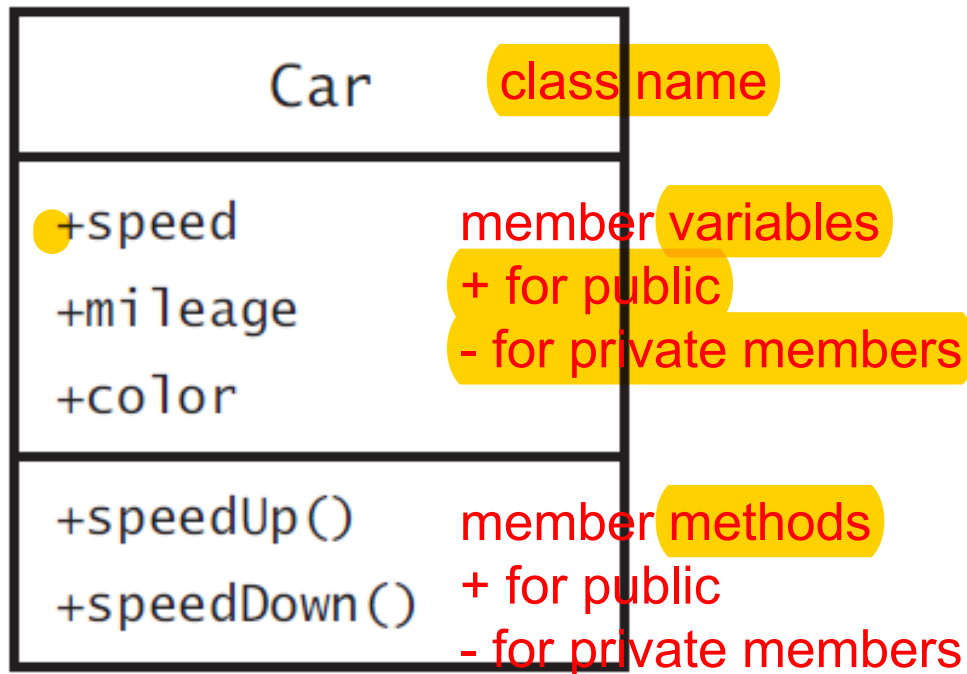
인수의 개수 : 5

2 3 4 5 6

인수의 개수 : 0

UML (Unified Modeling Language)

- class 정의를 diagram (box) 를 이용하여 표현



```
class Car {
    public int speed;
    public int mileage;
    public String color;
```

```
    public void speedUp();
    public void speedDown();
}
```

그림 8-7 . UML의 예

클래스와 클래스의 관계


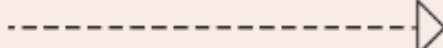
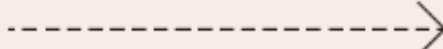



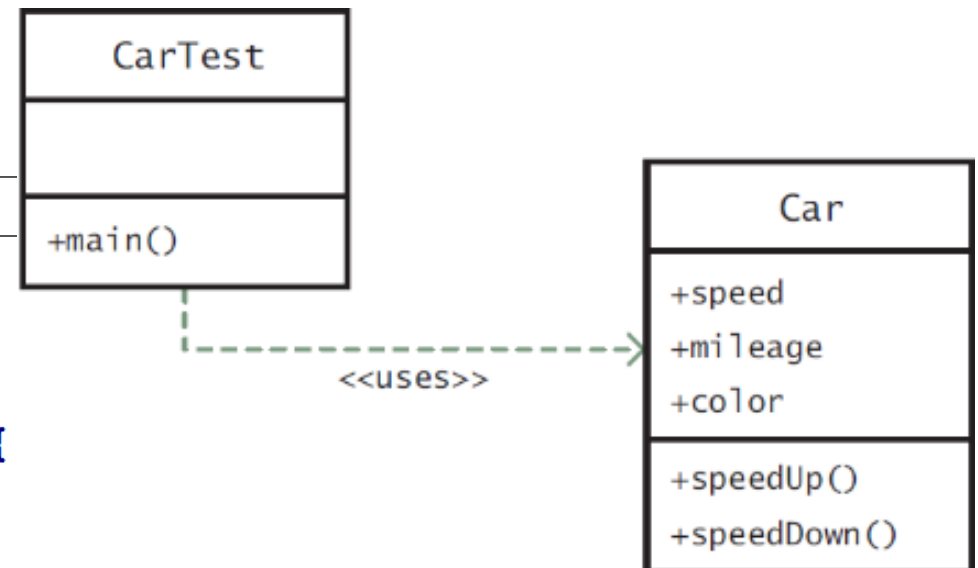
상속(inheritance)	
인터페이스 상속(interface inheritance)	
의존(dependency)	
집합(aggregation)	
연관(association)	
유형 연관(direct association)	

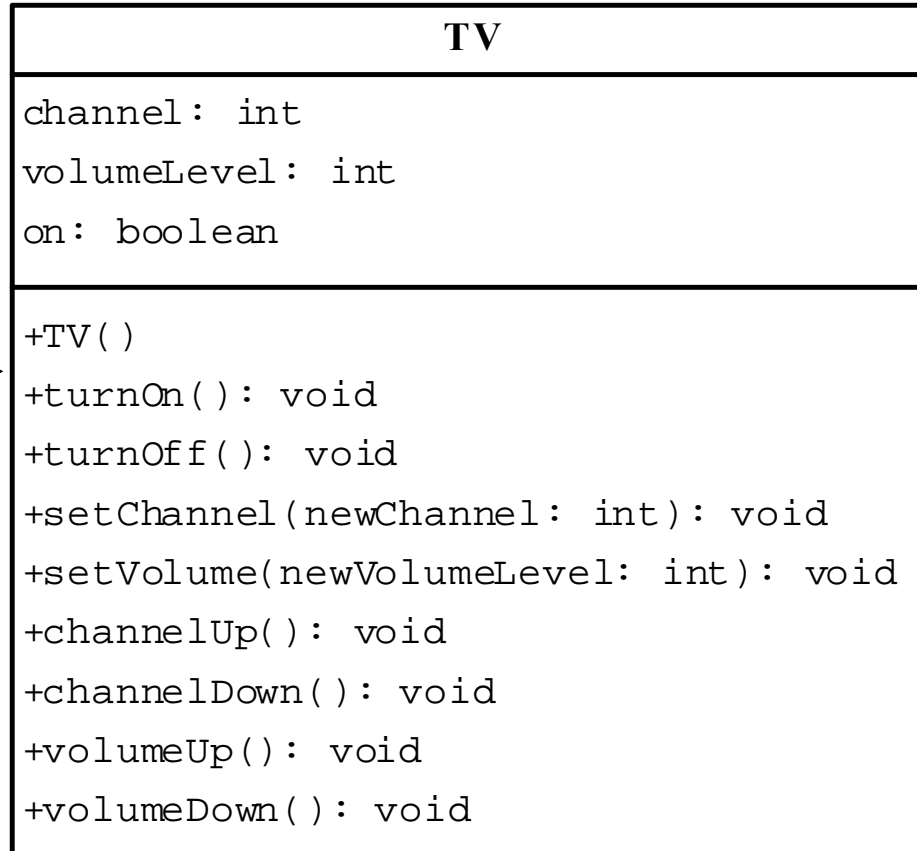
그림 8-8 . UML에서 사용되는 화살표의 종류

```
class Car{
    public int speed;
    public int mileage;
    public String color;
    public void speedUp() {
    public void speedDown() {
}
```

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



UML example for TV



The + sign indicates
a public modifier. →

The current channel (1 to 120) of this TV.
The current volume level (1 to 7) of this TV.
Indicates whether this TV is on/off.

Constructs a default TV object.
Turns on this TV.
Turns off this TV.
Sets a new channel for this TV.
Sets a new volume level for this TV.
Increases the channel number by 1.
Decreases the channel number by 1.
Increases the volume level by 1.
Decreases the volume level by 1.

TV Implementation from UML

```
public class TV {
    int channel = 1; //Default channel is 1
    int volumeLevel = 1; //Default volume level is 1
    boolean on = false; //By default TV is off

    public TV() { }
    public void turnOn() { on = true; }
    public void turnOff() { on = false; }
    public void setChannel(int newChannel) {
        if (on && newChannel >= 1 && newChannel <= 120)
            channel = newChannel;
    }
    public void setVolume(int newVolumeLevel) {
        if (on && newVolumeLevel >= 1 && newVolumeLevel <= 7)
            volumeLevel = newVolumeLevel;
    }
    public void channelUp() { if (on && channel < 120) channel++; }
    public void channelDown() { if (on && channel > 1) channel--; }
    public void volumeUp() { if (on && volumeLevel < 7) volumeLevel++; }
    public void volumeDown() { if (on && volumeLevel > 1) volumeLevel--; }
}
```

TV
channel: int volumeLevel: int on: boolean
+TV() +turnOn(): void +turnOff(): void +setChannel(newChannel: int): void +setVolume(newVolumeLevel: int): void +channelUp(): void +channelDown(): void +volumeUp(): void +volumeDown(): void

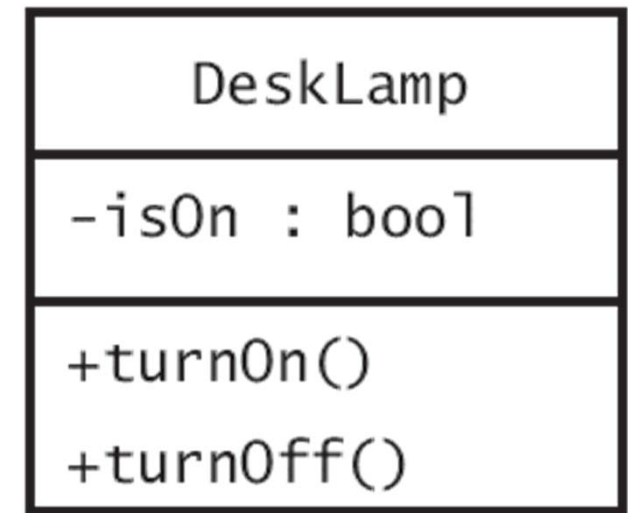


DeskLamp UML example

DeskLampTest.java

```
1 class DeskLamp{
2     //인스턴스 변수 정의
3     private boolean isOn;
4
5     //메서드 정의
6     public void turnOn(){ //램프 키는 메서드
7         isOn = true;
8     }
9     public void turnOff(){ //램프 끄는 메서드
10        isOn = false;
11    }
12    public String toString(){ //객체를 출력하면 이 메서드 호출
13        return "현재 상태는 " + (isOn==true?"켜짐":"꺼짐");
14    }
15 }
```

```
17 class DeskLampTest{
18     public static void main(String args []){
19         DeskLamp myLamp = new DeskLamp();
20         myLamp.turnOn();
21         System.out.println(myLamp); //객체를 출력??
22         System.out.println(myLamp.toString()); //위 문장과 완전히 같음
23         myLamp.turnOff();
24         System.out.println(myLamp);
25     }
26 }
```



U. WZUTU_JAVA_강의 17장
현재 상태는 켜짐
현재 상태는 켜짐
현재 상태는 꺼짐

Bank Account

BankAccountTest.java

```
01 class BankAccount {           // 은행 계좌
02     int accountNumber;         // 계좌 번호
03     String owner;              // 예금주
04     int balance;               // 잔액을
05
06     void deposit(int amount) {  // 저금
07         balance += amount;
08     }
09
10     void withdraw(int amount) { // 인출
11         balance -= amount;
12     }
13
14     public String toString(){
15         return "현재 잔액은 " + balance + "입니다.";
16     }
17 }
```

BankAccount

-owner : string
-accountNumber : int
-balance : int

+deposit()
+withdraw()

```
18 public class BankAccountTest {
19     public static void main(String[] args) {
20         BankAccount myAccount = new BankAccount();
21
22         // 객체의 메소드를 호출하려면 도트 연산자인 .을 사용한다.
23         myAccount.deposit(10000);
24         System.out.println(myAccount);
25         myAccount.withdraw(8000);
26         System.out.println(myAccount);
27
28     }
29 }
```

실행결과

현재 잔액은 10000입니다.
현재 잔액은 2000입니다.

Date

```
1 import java.util.Scanner;
2  /* DateTest.java */
3 class Date{
4     private int year;
5     private String month;
6     private int day;
7
8     public void setDate(int y, String m, int d){
9         year =y;
10        month =m;
11        day = d;
12    }
13    public void printDate(){
14        System.out.println(year+"년"+month+"월"+day+"일");
15    }
16    public int getYear(){return year;}
17    public void setYear(int y){year = y;}
18    //나머지 setter와 getter도 설정.....
19 }
21 class DateTest{
22     public static void main(String args []){
23         Date d = new Date();
24         d.printDate();//어떤 필드도 setting 되지 않은 상황
25         d.setDate(2014, "4월", 6);
26         d.printDate();
27         d.setYear(2015);
28         d.printDate();
29     }
```

Date
-year : int
-month : string
-day : int
+setDate()
+printDate()

0년null월0일
2014년4월6일
2015년4월6일

Constructors and Access Control

Power Java 2, Chapter 9

생성자

- 생성자(contructor)
 - 객체가 생성될 때에 필드에게 초기값을 제공하는 메서드
 - 클래스 이름과 같은 메서드
 - 반환형이 없음
 - 일반 메서드는 반환형을 반드시 기술해야 함
 - 접근 지정자는 public



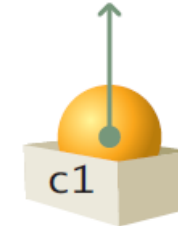
그림 9-1 . 생성자의 역할

CarTest.java

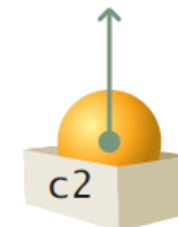
```
01 class Car {
02     private String color; // 색상
03     private int speed;    // 속도
04     private int gear;     // 기어
05     // 첫 번째 생성자
06
07     public Car(String c, int s, int g) {
08         color = c;
09         speed = s;
10         gear = g;
11     }
12
13     // 두 번째 생성자
14     public Car() {
15         color = "red";
16         speed = 0;
17         gear = 1;
18     }
19 }
20
21 public class CarTest {
22     public static void main(String args[]) {
23         Car c1 = new Car("blue", 100, 0); // 첫 번째 생성자 호출
24         Car c2 = new Car();                // 두 번째 생성자 호출
25     }
26 }
```



color	"blue"
speed	100
gear	1



color	"red"
speed	0
gear	1



생성자는 객체를 초기
화시키는 역할을 한다.

Default Constructor: 디폴트 생성자

- 만약 클래스 작성시에 생성자를 하나도 만들지 않는 경우에는 자동적으로 메소드의 몸체 부분이 비어있는 생성자가 만들어진다.

CarTest1.java

```
01 class Car {  
02     private String color;    // 색상  
03     private int speed;      // 속도  
04     private int gear;       // 기어  
05 }  
06 public class CarTest1 {  
07     public static void main(String args[]) {  
08         Car c1 = new Car();  // 디폴트 생성자 호출  
09     }  
10 }
```

← 컴파일러가 디폴트 생성자를
자동으로 만든다.

```
1 class Car{  
2     String color;  
3     int gear;  
4     int speed;  
5     public Car(){ }  
6 }
```

주의할 점

- 생성자가 하나라도 정의되어 있으면 디폴트 생성자는 만들어지지 않는다.

CarTest2.java

```
01 class Car {  
02     private String color;    // 색상  
03     private int speed;      // 속도  
04     private int gear;       // 기어  
05     public Car(String c, int s, int g) {  
06         color = c;  
07         speed = s;  
08         gear = g;  
09     }  
10 }  
11 public class CarTest2 {  
12     public static void main(String args[]) {  
13         Car c1 = new Car();    // 오류!  
14     }  
15 }
```

생성자가 하나라도 선언되면 디폴트
생성자는 만들지 않는다.




생성자에서 메소드 호출

- `this()`는 생성자를 호출한다.

Car.java

```
01 public class Car {  
02     private int speed;    // 속도  
03     private int gear;    // 기어  
04     private String color; // 색상  
05  
06     // 첫 번째 생성자  
07     public Car(String c, int s, int g) {  
08         color = c;  
09         speed = s;  
10         gear = g;  
11     }  
12     // 색상만 주어진 생성자  
13     public Car(String c) {  
14         this(c, 0, 1);    // 첫 번째 생성자를 호출한다.  
15     }  
16 }
```



```
01 import java.util.Scanner;
```

```
02
```

```
03 class Date {
```

```
04     private int year;
```

```
05     private String month;
```

```
06     private int day;
```

```
07
```

```
08     public Date() {
```

```
09         month = "1월";
```

```
10         day = 1;
```

```
11         year = 2009;
```

```
12     }
```

```
13
```

```
14     public Date(int year, String month, int day) { // 생성자
```

```
15         setDate(year, month, day);
```

```
16     }
```

```
17
```

```
18     public Date(int year) { // 생성자
```

```
19         setDate(year, "1월", 1);
```

```
20     }
```

```
21
```

```
22     public void setDate(int year, String month, int day) {
```

```
23         this.month = month;
```

```
24         this.day = day;
```

```
25         this.year = year;
```

```
26     }
```

```
27 }
```

```
28 public class DateTest {
```

```
29
```

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

```
31         Date date1=new Date(2009,"3월", 2);
```

```
32         Date date2=new Date(2010);
```

```
33         Date date3=new Date();
```

```
34     }
```

```
35 }
```

// 기본 생성자

생성자 중복 정의

// 생성자

// 생성자

// this는 현재 객체를 가리킨다

예제 #1

DateTest.java

예제 #2

TimeTest.java

```
01 class Time {
02     private int hour;    // 0 - 23
03     private int minute;  // 0 - 59
04     private int second;  // 0 - 59
05
06     // 첫 번째 생성자
07     public Time() {
08         this(0, 0, 0);
09     }
10
11     // 두 번째 생성자
12     public Time(int h, int m, int s) {
13         setTime(h, m, s);
14     }
15
16     // 시간 설정 함수
17     public void setTime(int h, int m, int s) {
18         hour = ((h >= 0 && h < 24) ? h : 0);    // 시간 검증
19         minute = ((m >= 0 && m < 60) ? m : 0);  // 분 검증
20         second = ((s >= 0 && s < 60) ? s : 0);  // 초 검증
21     }
22
23     // "시:분:초"의 형식으로 출력
24     public String toString() {
25         return String.format("%02d:%02d:%02d", hour, minute, second);
26     }
27 }
```

```
28 public class TimeTest {
29     public static void main(String args[]) {
30         // Time 객체를 생성하고 초기화한다.
31         Time time = new Time();
32
33         System.out.print("기본 생성자 호출 후 시간: ");
34         System.out.println(time.toString());
35
36         // 두 번째 생성자 호출
37         Time time2 = new Time(13, 27, 6);
38         System.out.print("두번째 생성자 호출 후 시간: ");
39         System.out.println(time2.toString());
40
41         // 올바르지 않은 시간으로 설정해본다.
42         Time time3 = new Time(99, 66, 77);
43         System.out.print("올바르지 않은 시간 설정 후 시간: ");
44         System.out.println(time3.toString());
45     }
46 }
```

실행결과

기본 생성자 호출 후 시간: 00:00:00
두번째 생성자 호출 후 시간: 13:27:06
올바르지 않은 시간 설정 후 시간: 00:00:00

```

01 class Point {
02     private int x, y;
03     // 생성자
04     public Point(int a, int b) {
05         x = a;
06         y = b;
07     }
08 }

```

Circle 객체가 Point 객체를 포함하고 있다.

```

10 class Circle {
11     private int radius = 0;
12     private Point center; // Point 참조 변수가 필드로 선언되어 있다.
13
14     // 생성자
15     public Circle(Point p, int r) {
16         center = p;
17         radius = r;
18     }
19 }

```

예제 #3

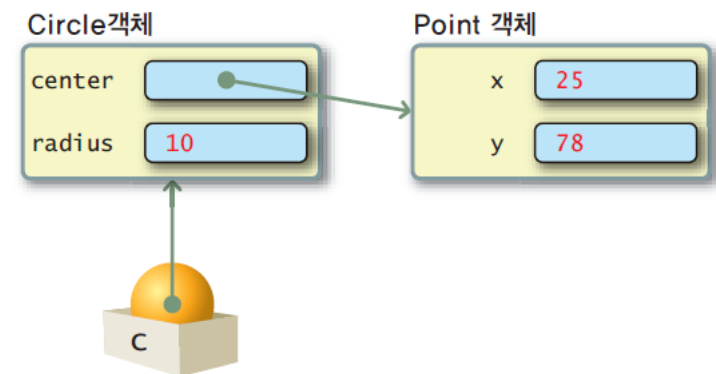
CircleTest.java

```

20
21 public class CircleTest {
22     public static void main(String args[]) {
23         // Circle 객체를 생성하고 초기화한다.
24         Point p = new Point(25, 78);
25         Circle c = new Circle(p, 10);
26     }
27 }

```

Circle 객체를 생성할 때,
Point 객체 참조값을 넘긴다.



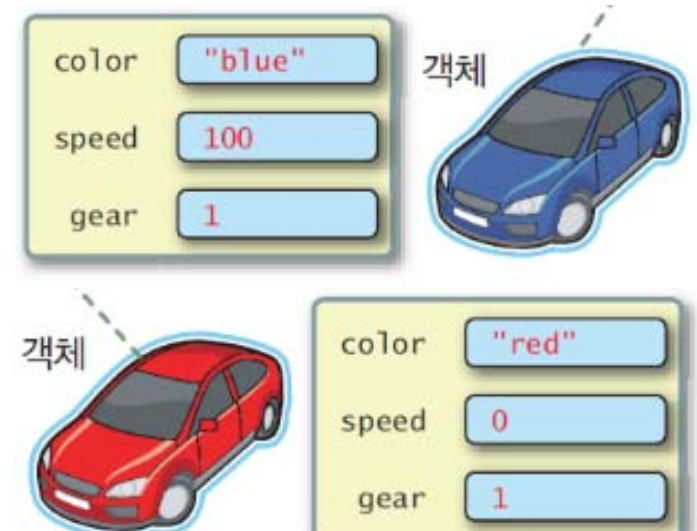
중간 점검 문제

1. 만약 클래스 이름이 MyClass라면 생성자의 이름은 무엇인가?
 - Myclass
2. 생성자의 반환형은?
 - 없다
3. 생성자안에서 this()의 의미는?
 - 인자 없는 생성자 호출

정적 변수

```
class Car{  
    int speed;  
    int gear;  
    String color;  
}
```

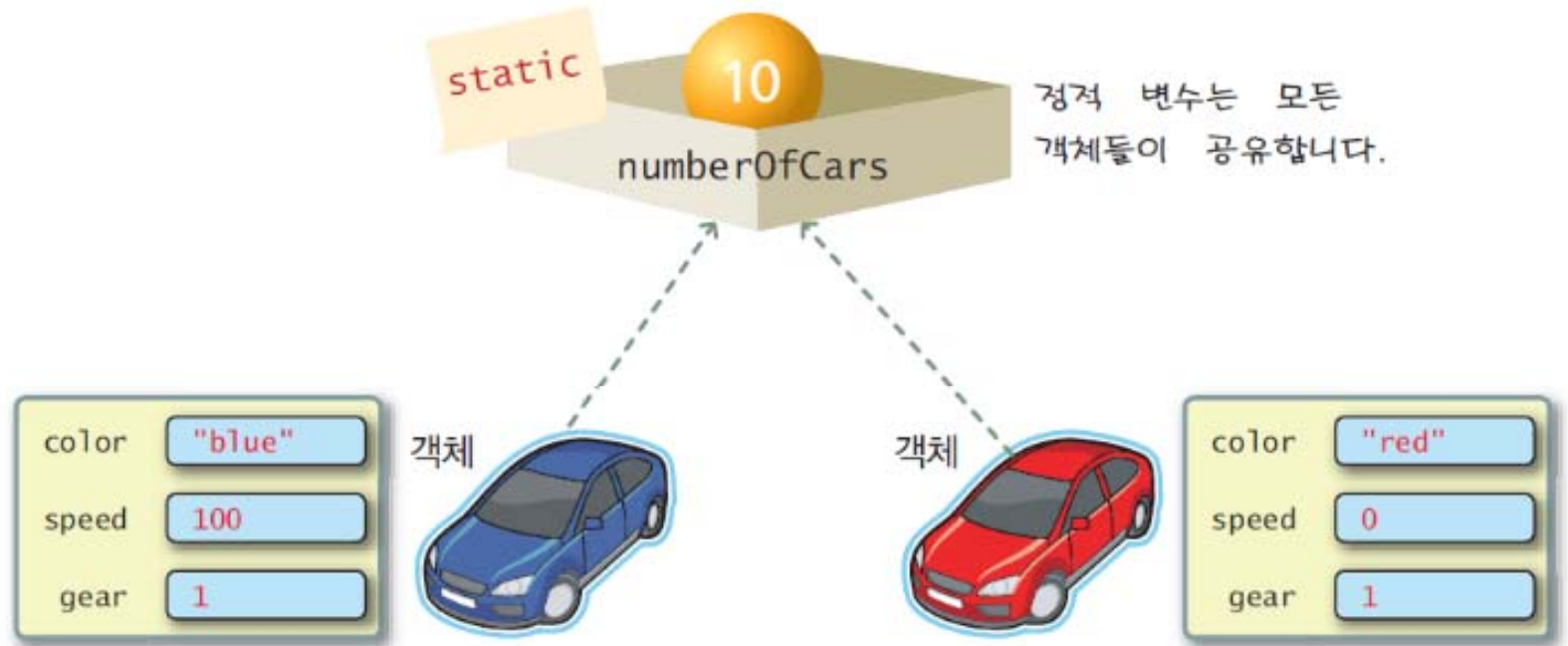
```
class CarTest{  
    public static void main(String a []){  
        Car myCar = new Car();  
        myCar.color="blue";  
        myCar.speed=100;  
        myCar.gear=1;  
        Car yourCar = new Car();  
        yourCar.color="Red";  
        yourCar.speed=0;  
        yourCar.gear=1;  
    }  
}
```



Math Class

정적 변수

- 인스턴스 변수(instance variable): 객체마다 하나씩 있는 변수
- 정적 변수(static variable): 모든 객체를 통틀어서 하나만 있는 변수



Car.java

```
01 public class Car {
02     private String color;
03     private int speed;
04     private int gear;
05     // 자동차의 시리얼 번호
06     private int id;
07     private static int numberOfCars = 0; ← 정적 변수
08
09     public Car(String c, int s, int g) {
10         color = c;
11         speed = s;
12         gear = g;
13         // 자동차의 개수를 증가하고 id 번호를 할당한다.
14         id = ++numberOfCars;
15     }
16 }
```

- 정적 변수를 외부에서 사용하기 위해서는
 - 클래스이름.정적변수
 - 객체이름.정적변수가 아님!!!

```
int n = Car.numberOfCars ;
```

정적 메소드

- 정적 메소드(static method)
 - 객체를 생성하지 않고 사용할 수 있는 메소드
- 예) Math 클래스에 들어 있는 각종 수학 메소드들

```
double value = Math.sqrt(9.0);
```

- 정적 메소스에서 인스턴스 메소드를 호출할 수 없음.
Why?

```
public class Test {  
    public static void main(String args[]) {  
        add(10,20);    // 오류!! 정적 메소드 안에서 인스턴스 메소드 호출  
    }  
    int add(int x, int y) {  
        return x + y;  
    }  
}
```

객체가 생성되지 않았으므로
인스턴스 메소드는 존재하지 않는다.

CarTest3.java

```

01 class Car {
02     private String color;
03     private int speed;
04     private int gear;
05     // 자동차의 시리얼 번호
06     private int id;
07     // 실체화된 Car 객체의 개수를 위한 정적 변수
08     private static int numberOfCars = 0;
09
10     public Car(String c, int s, int g) {
11         color = c;
12         speed = s;
13         gear = g;
14         // 자동차의 개수를 증가하고 id 번호를 할당한다.
15         id = ++numberOfCars;
16     }
17     // 정적 메소드
18     public static int getNumberOfCars() {
19         return numberOfCars; // OK!
20     }
21 }
22 public class CarTest3 {
23     public static void main(String args[]) {
24         Car c1 = new Car("blue", 100, 1); // 첫 번째 생성자 호출
25         Car c2 = new Car("white", 0, 1); // 첫 번째 생성자 호출
26         int n = Car.getNumberOfCars(); // 정적 메소드 호출
27         System.out.println("지금까지 생성된 자동차 수 = " + n);
28     }

```

정적 메소드에서는 인스턴스 변수와
인스턴스 메소드에 접근할 수 없다.

상수

- 상수는 공간을 절약하기 위하여 정적 변수로 선언된다.

```
public class Car {
```

```
...
```

```
static final int MAX_SPEED = 350;
```

```
...
```

```
}
```

← 상수는 정적 변수로 만들어서 공유하는 것이 메모리 공간을 절약한다.

```

01 import java.util.*;
02
03 class Employee {
04     private String name;
05     private double salary;
06
07     private static int count = 0;    // 정적 변수
08
09     // 생성자
10     public Employee(String n, double s) {
11         name = n;
12         salary = s;
13         count++; // 정적 변수인 count를 증가
14     }
15
16     // 객체가 소멸될 때 호출된다.
17     protected void finalize() {
18         count--; // 직원이 하나 줄어드는 것이므로 count를 하나 감소
19     }
20
21     // 정적 메소드
22     public static int getCount() {
23         return count;
24     }
25 }

```

```

27 public class EmployeeTest {
28     public static void main(String[] args) {
29         Employee e1,e2,e3;
30         e1 = new Employee("김철수", 35000);
31         e2 = new Employee("최수철", 50000);
32         e3 = new Employee("김철호", 20000);
33
34         int n = Employee.getCount();
35         System.out.println("현재의 직원수=" + n);
36     }
37 }

```

정적변수의 예

객체가 소멸될 때 호출된다.

중간 점검 문제

1. 정적 변수는 어떤 경우에 사용하면 좋은가?
 - 클래스의 모든 객체들에 의해 공유될 때
2. 정적 변수나 정적 메소드를 사용할 때, 클래스 이름을 통하여 접근하는 이유는 무엇인가?
 - 정적 변수와 정적 메소드는 객체를 생성할 필요가 없고 매개 변수를 통하여 전달된 값만 있으면 되므로 클래스 이름을 통하여 접근
3. main() 안에서 인스턴스 메소드를 호출할 수 없는 이유는 무엇인가?
 - main()메소드도 정적 메소드이기 때문

this 참조

- 자기 자신을 참조하는 키워드

```
public void setSpeed(int speed)
{
    this.speed = speed;
}
```

필드 speed와 매개변수 speed를
구별하기 위하여 this 사용

// speed는 매개변수, this.speed는 필드

- 생성자를 호출할 때도 사용된다.

```
// 두 번째 생성자
public Time(int h, int m, int s) {
    this.setTime(h, m, s); // this는 없어도 된다.
}
```

PersonTest.java

```
01 class Person {
02     private String lastName;
03     private String firstName;
04
05     String getLastName() {
06         return lastName;
07     }
08
09     String getFirstName() {
10         return firstName;
11     }
12
13     public Person(String lastName, String firstName) {
14         this.lastName = lastName;           // this는 현재 객체를 가리킨다.
15         this.firstName = firstName;         // this는 현재 객체를 가리킨다.
16     }
17
18     public String buildName() {
19         return String.format("%s %s\n", this.getLastName(), getFirstName()); // ❶
20     }
21 }
22 public class PersonTest {
23     public static void main(String args[]) {
24         Person person = new Person("홍", "길동");
25         System.out.println(person.buildName());
26     }
27 }
```

필드

매개변수

중간 점검 문제



1. 필드의 경우, private로 만드는 것이 바람직한 이유는 무엇인가?
 - 필드를 다른 클래스가 직접 사용하지 못하게 하기 위해서
2. 필드를 정의할 때 아무런 접근 제어 수식자를 붙이지 않으면 어떻게 되는가?
 - 디폴트 package가 된다
3. this의 주된 용도는 무엇인가?
 - 자기 자신을 참조하는데 사용
4. this()와 같이 표기하면 무엇을 의미하는가??
 - 자신의 생성자 호출

클래스 A의 메소드에서 클래스 B의 메소드들을 호출

Complex.java

```
01 public class Complex {
02     private double real;
03     private double imag;
04
05     public Complex(double r, double i) {
06         real = r;
07         imag = i;
08     }
09     ...
10     double getReal() {
11         return real;
12     }
13     double getImag() {
14         return imag;
15     }
16     public Complex add(Complex c) {
17         double resultReal = real + c.getReal();
18         double resultImag = real + c.getImag();
19         return new Complex(resultReal, resultImag);
20     }
21 }
```

복소수를 표현한다.

객체 참조값을 매개변수로 받는다.

객체가 생성되고 객체의 참조값이 반환된다.