

제4장 클래스와 객체

2018년1학기

컴퓨터공학과 김 명 교수

목차

- 객체지향적 언어의 목적과 특징
- 클래스 선언과 객체의 생성
- 메소드와 생성자
- 메소드 인자 전달 방식
- 메소드 오버로딩
- this 레퍼런스
- 객체의 치환
- this()
- 객체의 소멸과 garbage collection
- 멤버 접근 지정자
- static 멤버와 non-static 멤버

객체지향적 언어의 목적

3

□ 소프트웨어 생산성 향상

- 컴퓨터 산업 발전에 따른 소프트웨어의 생명 주기 (life cycle) 단축됨
- 객체 지향적 언어는 상속, 다형성, 캡슐화 등, 소프트웨어 재사용을 위한 여러 장치를 내장하고 있음
- 소프트웨어의 재사용과 부분 수정을 통해 소프트웨어를 다시 만드는 부담을 대폭 줄임으로써 소프트웨어 생산성을 향상시킴

□ 실세계에 대한 쉬운 모델링

- 과거 : 수학 계산, 통계 처리 등 처리 과정, 계산 절차가 중요
- 현재 : 실세계에서 발생하는 일을 프로그래밍 (→ 객체지향)

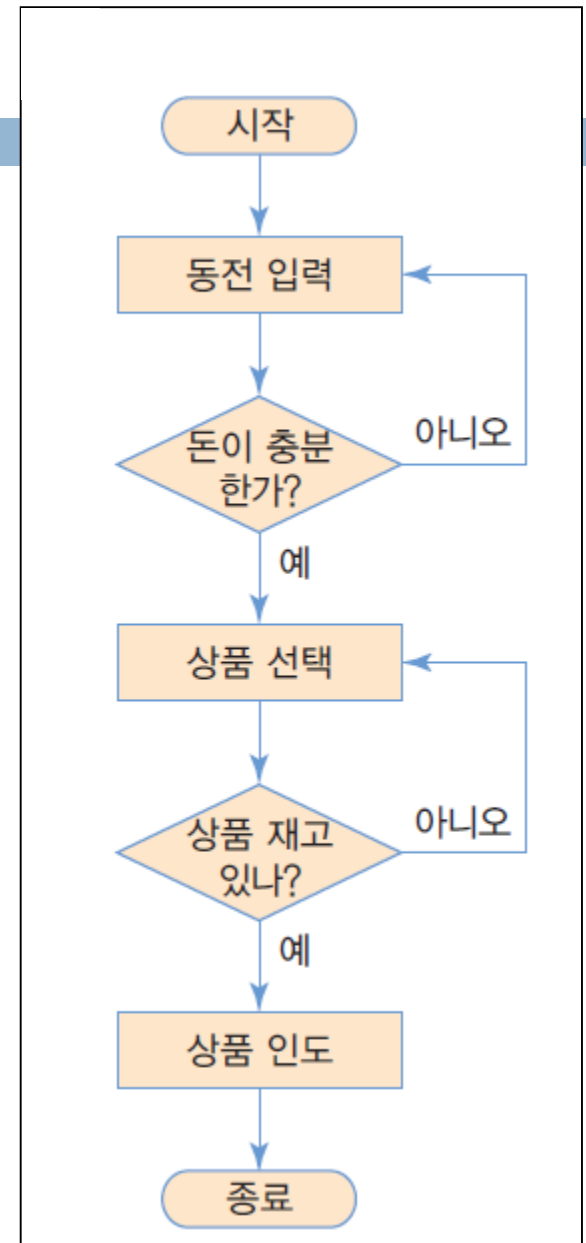
절차지향적 프로그래밍

4

- 음료수 자동판매기를 위한 "절차 지향적" 프로그램의 실행 과정



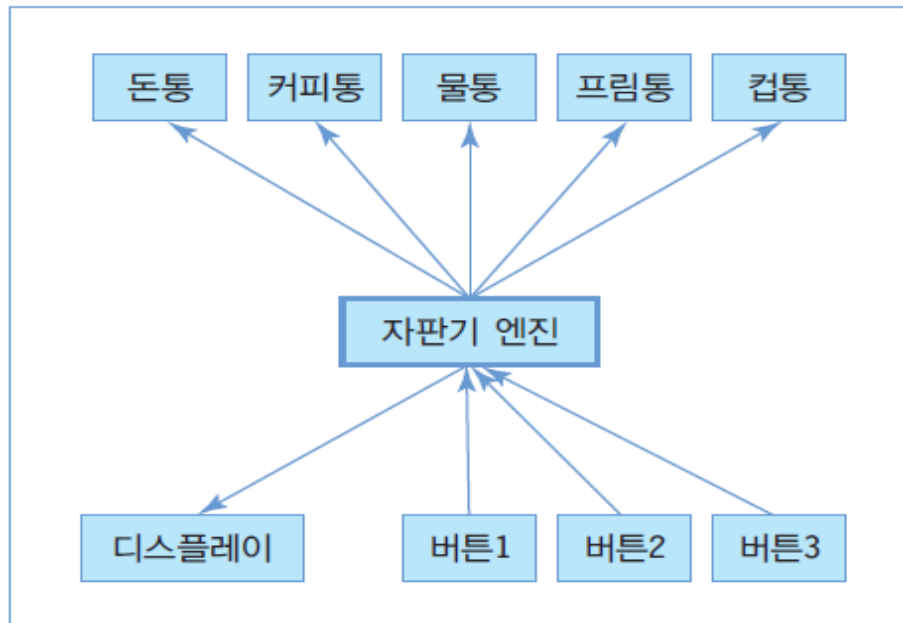
- 작업 순서를 표현하는 컴퓨터 명령 집합
- 함수들의 집합으로 프로그램 작성



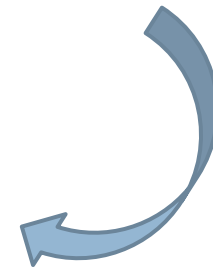
객체 지향적 프로그래밍

5

- ▣ 프로그램을 실제 세상에 가깝게 **모델링**
- ▣ 컴퓨터가 수행하는 작업을 객체들간의 상호 작용으로 표현
- ▣ 클래스 또는 객체들의 집합으로 프로그램 작성



객체지향적 프로그래밍의
객체들의 상호 관련성



커피 자동판매기

객체 지향 언어의 특성 : 캡슐화 (Encapsulation)

6

- 클래스 내에 데이터와 메소드(method, 함수)를 정의하고 구현
- 외부에서는 공개된 메소드를 호출하여 해당 객체에 접근
- 객체 내의 데이터에 대한 보안, 보호, 외부 접근 제한 가능

실세계의 캡슐화



TV



자판기



카메라

자바 객체의 캡슐화

객체

```
String name;  
int age;
```

데이터 필드(field)

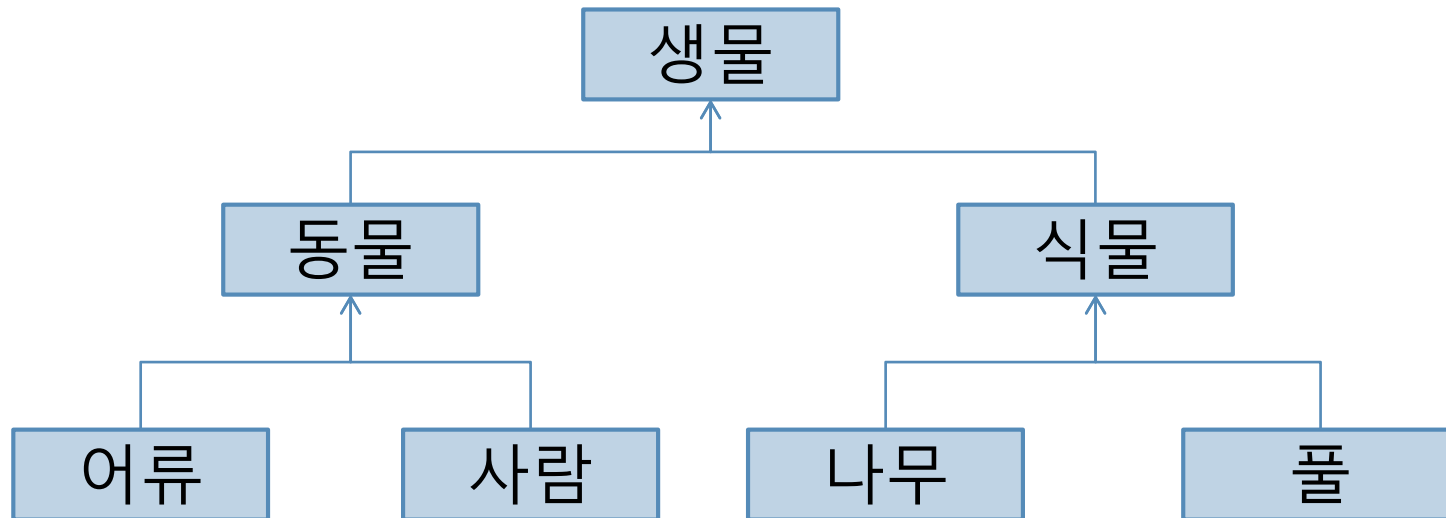
```
void speak( );  
void eat( );  
void study( );
```

메소드(method)

객체 지향 언어의 특성 : 상속 (Inheritance)

7

실세계에서의 상속 - 유전적인 상속 관계 표현



객체 지향 언어의 특성 : 상속 (Inheritance)

8

Animal 객체 (호돌이)

```
class Animal {  
    String name;  
    int age;  
    void eat( ) {...}  
    void sleep( ) {...}  
    void love( ) {...}  
}
```

```
String name;  
int age;  
void eat( );  
void sleep( );  
void love( );
```

Human 객체 (최이화)

```
class Human extends  
Animal {  
    String hobby;  
    String job;  
    void work( ) {...}  
    void cry( ) {...}  
    void laugh( ) {...}  
}
```

상속

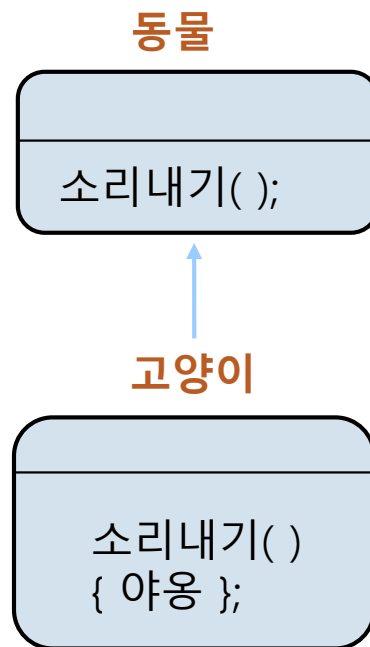
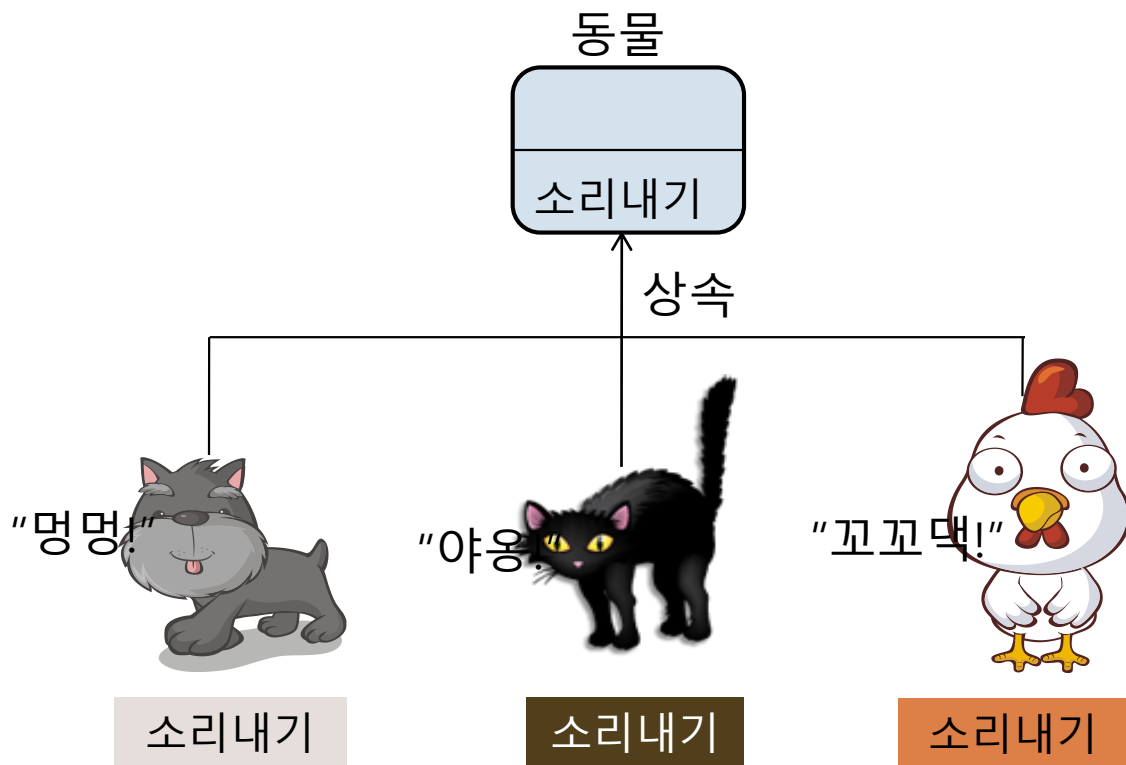
```
String name;  
int age;  
void eat ( );  
void sleep( );  
void love( );  
String hobby;  
String job;  
void work( );  
void cry( );  
void laugh( );
```

- 상속 : 상위 클래스의 특성을 하위 클래스가 물려받음
 - 상위 클래스 : 슈퍼 클래스
 - 하위 클래스 : 서브 클래스
- 서브 클래스
 - 슈퍼 클래스 **코드의 재사용**
 - 새로운 특성 추가 가능
- 다중 상속을 지원 않음
 - 인터페이스를 통해 다중 상속과 같은 효과 얻음

객체 지향 언어의 특성 : 다형성 (Polymorphism)

9

- 같은 메소드 (또는 함수) 가 객체에 따라 다른 동작을 수행한다.
- 다형성은 오버라이딩과 밀접한 관계가 있다.



클래스와 객체

10

□ 클래스 (class)

- 객체의 공통된 특징을 기술
- 객체의 특성과 행위를 정의

□ 객체 (object)

- 클래스의 인스턴스 (instance, 실체)
- 물리적 공간을 갖는 구체적인 것, 실체

□ 사례

- | | |
|----------------|---------------------|
| □ 클래스: 소나타 자동차 | 객체: 출고된 실제 소나타 100대 |
| □ 클래스: 벽시계 | 객체: 우리집 벽에 걸린 벽시계들 |
| □ 클래스: 책상 | 객체: 우리가 사용중인 실제 책상들 |

사람을 사례로 든 클래스와 객체 사례

11

클래스: Person

이름, 직업, 나이, 성별, 혈액형

밥 먹기, 잠자기, 말하기, 걷기



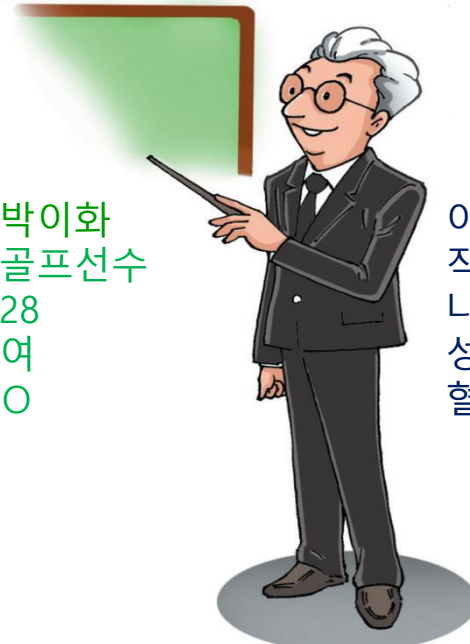
이름 김이화
직업 의사
나이 45
성별 여
혈액형 A

객체 : 김이화



이름 박이화
직업 골프선수
나이 28
성별 여
혈액형 O

객체 : 박이화



이름 나이화
직업 교수
나이 47
성별 남
혈액형 AB

객체: 나이화

클래스 구성

12

클래스
접근 권한

클래스
키워드

클래스
이름

```
public class Person {  
    public String name;  
    public int age;
```

필드(field)

```
    public Person() {  
    }
```

```
    public Person(String s) {  
        name = s;  
    }
```

```
    public String getName() {  
        return name;  
    }
```

```
}
```

Class Person

name



age



Person()

Person(String s)

getName()

생성자
(constructor)

생성자

메소드
(method)

클래스 선언

13

□ 클래스 접근 권한, public

- public 접근 권한은 다른 모든 클래스들이 이 클래스에 접근 가능함을 의미

□ class Person

- Person 이라는 이름의 클래스 정의
- 클래스는 {로 시작하여 }로 닫으며 이곳에 모든 멤버 필드와 메소드를 구현

□ 필드(field)

- 값을 저장할 멤버 변수들
- 필드 앞에 붙은 접근 지정자 public은 이 필드가 다른 클래스에서 접근될 수 있도록 공개한다는 의미

□ 생성자(constructor)

- 클래스의 이름과 동일한 메소드
- 클래스의 객체가 생성될 때만 호출되는 메소드

□ 메소드(method)

- 메소드는 함수이며, 객체의 행위를 구현
- 메소드 앞에 붙은 접근 지정자 public은 이 메소드가 다른 클래스에서 접근될 수 있도록 공개한다는 의미

객체 생성

14

□ 객체 생성

- ▣ new 키워드를 이용하여 생성한다. new는 객체의 생성자를 호출한다.

□ 객체 생성 과정

- ▣ 객체에 대한 레퍼런스 변수 선언
- ▣ 객체 생성

```
public class PersonDemo1 {  
    public static void main(String[] args) {  
        Person aPerson; // 레퍼런스 변수 aPerson 선언  
        aPerson = new Person("김이화"); // Person 객체 생성  
        aPerson.age = 30; // 객체 멤버 age에 저장  
        int i = aPerson.age; // 객체 멤버 age 읽음  
        String s = aPerson.getName(); // 객체 멤버 메소드 호출  
        System.out.println(s); // s 값 출력  
    }  
}
```

객체 생성 및 사용 예

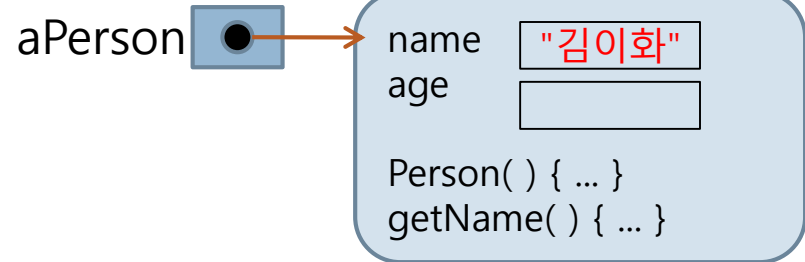
15

(1) Person aPerson;

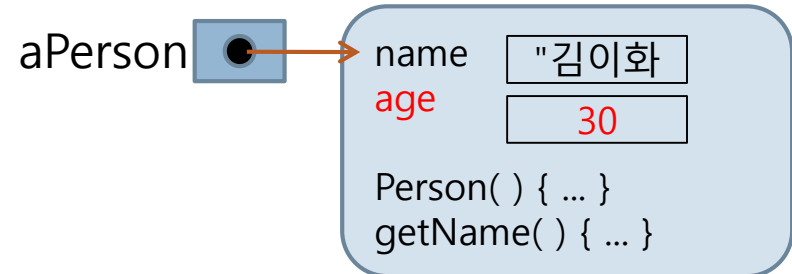
aPerson 

Person 타입의 객체

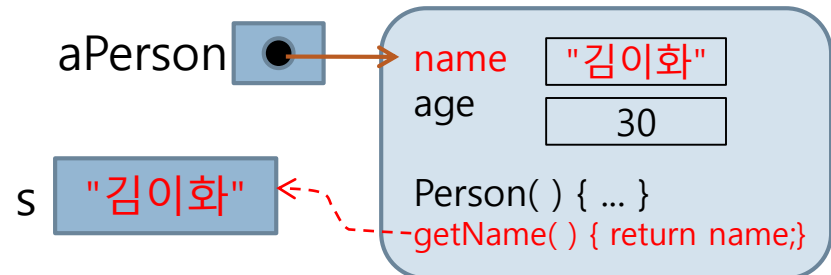
(2) aPerson = new Person("김이화");



(3) aPerson.age = 30;



(4) String s = aPerson.getName();



객체의 활용

16

□ 객체의 멤버 접근

객체레퍼런스 . 멤버

```
public class PersonDemo1 {  
    public static void main(String[] args) {  
        Person aPerson; // 레퍼런스 변수 aPerson 선언  
        aPerson = new Person("김이화"); // Person 객체 생성  
        aPerson.age = 30; // 객체 멤버 age에 저장  
        int i = aPerson.age; // 객체 멤버 age 읽음  
        String s = aPerson.getName(); // 객체 멤버 메소드 호출  
        System.out.println(s); // s 값 출력  
    }  
}
```

객체의
필드에
값 대입

객체의 필드에서
값 읽기

객체의
메소드 호출

(예제) 상품 1개를 표현하는 클래스 Goods

17

```
public class Goods {  
    String name;  
    int price;  
    int numberOfStock;  
    int sold;  
}  
  
public class GoodsDemo {  
    public static void main(String[] args) {  
        Goods camera = new Goods();  
        camera.name = "Nikon";  
        camera.price = 400000;  
        camera.numberOfStock = 30;  
        camera.sold = 50;  
  
        System.out.println("상품 이름:" + camera.name);  
        System.out.println("상품 가격:" + camera.price);  
        System.out.println("재고 수량:" + camera.numberOfStock);  
        System.out.println("팔린 수량:" + camera.sold);  
    }  
}
```

클래스 Goods

* 스트링 1개와 정수 3개 저장

main() 메소드

* Goods 객체 생성

- 객체 이름 : camera

- 객체 필드 값

"Nikon", 가격:400000원,
재고 30개, 팔린 개수 50개

* 설정된 값을 화면에 출력

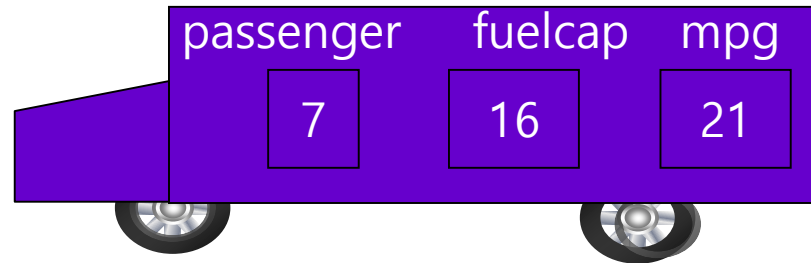
상품 이름:Nikon
상품 가격:400000
재고 수량:30
팔린 수량:50

실행 결과

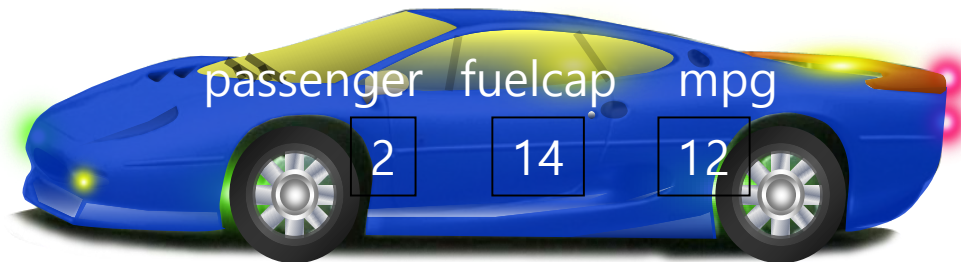
(예제) 차량 연비 계산 프로그램 (1/5)

18

minivan



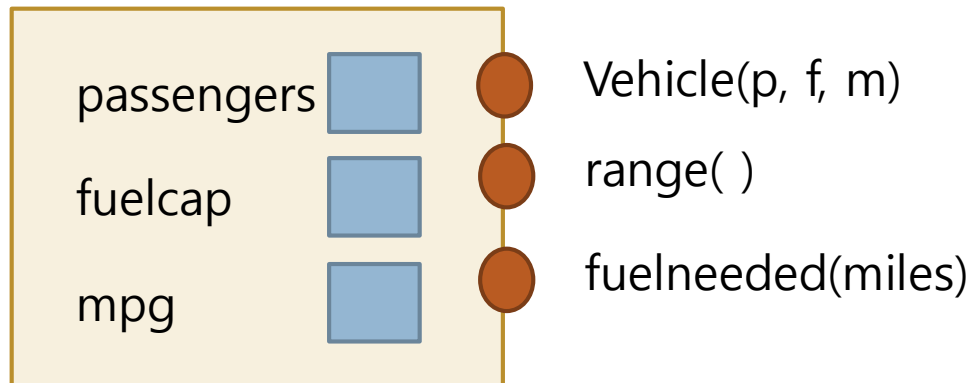
sportscar



Vehicle 클래스 (2/5)

19

```
public class Vehicle {  
    public int passengers; // 승객 수  
    public int fuelcap;    // 개스 탱크 크기  
    public int mpg;        // 연비  
  
    // Vehicle 클래스의 생성자  
    public Vehicle(int p, int f, int m) {  
        passengers = p;  
        fuelcap = f;  
        mpg = m;  
    }  
}
```

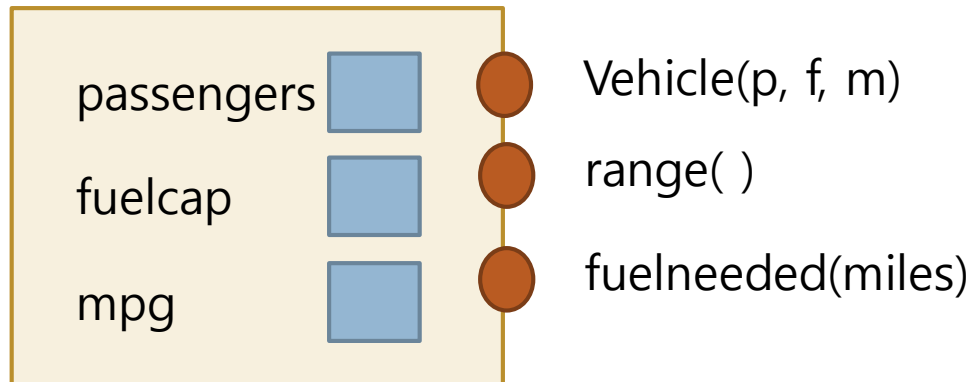


Vehicle 클래스 (3/5)

20

```
// 주어진 연료로 주행할 수 있는 거리  
public int range() {  
    return mpg * fuelcap;  
}
```

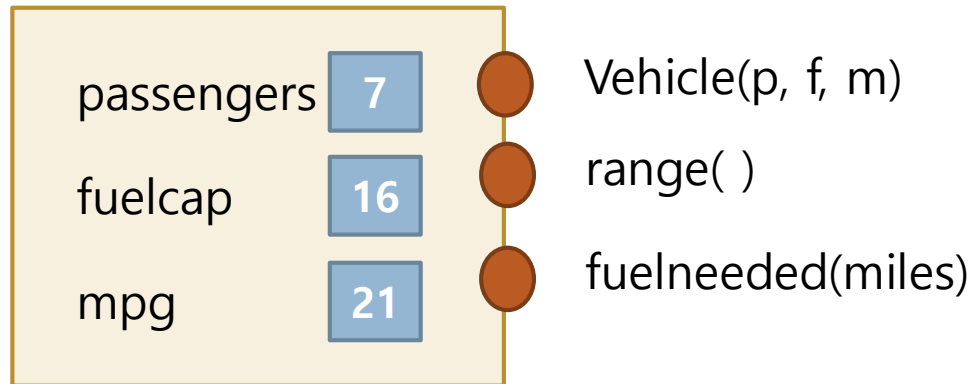
```
// 주어진 거리를 주행하기 위한 연료량  
public double fuelneeded(int miles) {  
    return (double) miles / mpg;  
}  
}
```



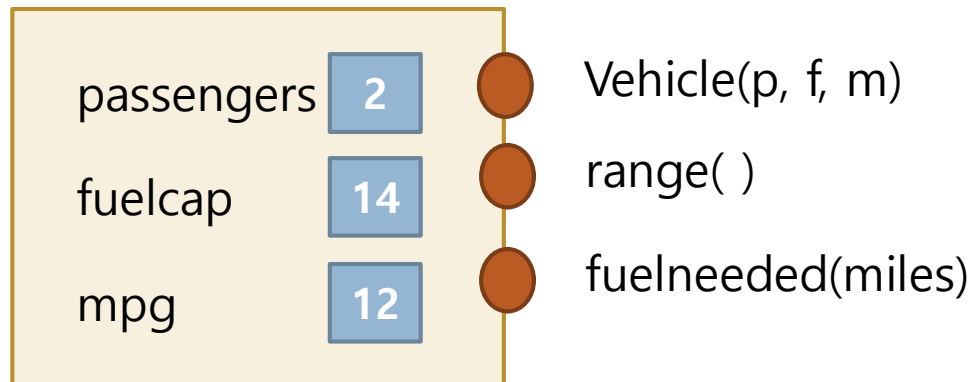
Vehicle 클래스 객체 만들기 (4/5)

21

minivan



sportscar



VehicleDemo 프로그램 (5/5)

22

```
public class VehicleDemo1 {  
    public static void main(String[] args) {  
        // construct complete vehicles  
        Vehicle minivan    = new Vehicle(7, 16, 21);  
        Vehicle sportscar = new Vehicle(2, 14, 12);  
        double gallons;  
        int dist = 252;  
        gallons = minivan.fuelneeded(dist);  
        System.out.println("To go " + dist + " miles minivan needs " +  
                           gallons + "gallons of fuel.");  
        gallons = sportscar.fuelneeded(dist);  
        System.out.println("To go " + dist + " miles sportscar needs " +  
                           gallons + " gallons of fuel.");  
    }  
}
```

실행결과

```
To go 252 miles minivan needs 12.0gallons of fuel.  
To go 252 miles sportscar needs 21.0 gallons of fuel.
```

객체 배열 생성 및 사용 (1/2)

23

```
Person[] pa;
pa = new Person[10];
for(int i=0; i<pa.length; i++) {
    pa[i] = new Person();
    pa[i].age = 30 + i;
}
```

배열에 대한 레퍼런스 선언

레퍼런스 배열 생성

배열의 원소 객체 생성

객체 배열 사용

```
for(int i=0; i<pa.length; i++) // 배열 pa의 모든 원소 객체의 age를 출력한다.
    System.out.print(pa[i].age + " ");
```

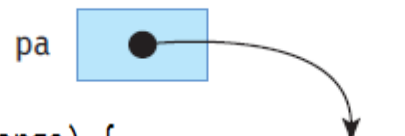
객체 배열 생성 및 사용 (2/2)

24

Person[] pa;

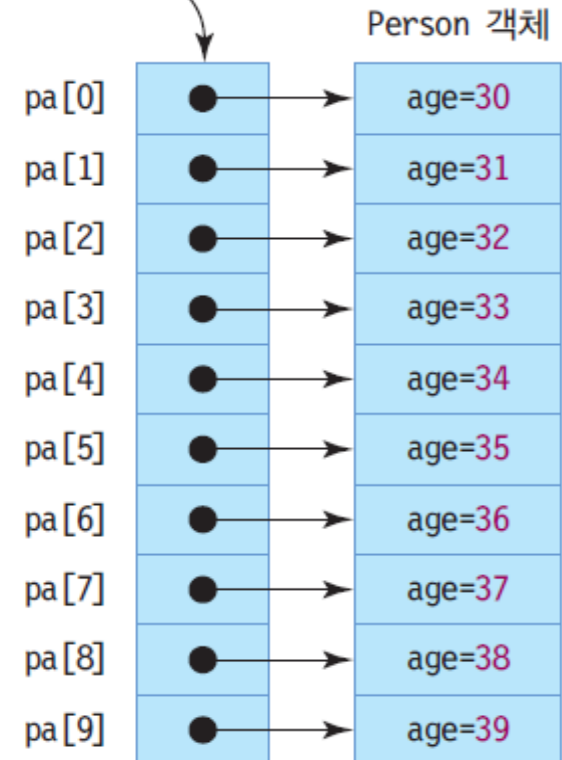
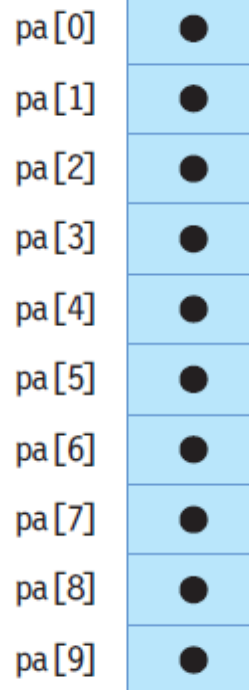
pa = new Person[10];

```
for(int i=0; i<pa.length; i++) {  
    pa[i] = new Person();  
    pa[i].age = 30 + i;  
}
```



```
public static void main(String [] args) {  
    Person[] pa;  
    pa = new Person[10];  
    for (int i=0;i<pa.length;i++) {  
        pa[i] = new Person();  
        pa[i].age = 30 + i;  
    }  
  
    for (int i=0;i<pa.length;i++)  
        System.out.print(pa[i].age+ " ");  
}
```

30 31 32 33 34 35 36 37 38 39



상품 객체 배열 생성 및 사용 (1/2)

25

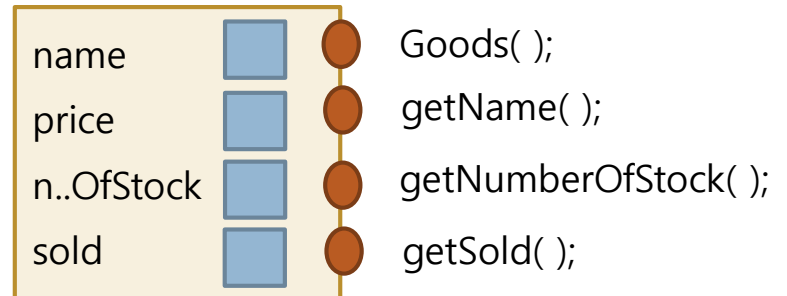
```
public class Goods {  
    private String name;  
    private int price;  
    private int numberOfStock;  
    private int sold;
```

상품 (Goods) 클래스 정의

```
    public Goods(String name, int price, int numberOfStock, int sold) {  
        this.name = name;  
        this.price = price;  
        this.numberOfStock = numberOfStock;  
        this.sold = sold;
```

```
    }  
    public String getName() {return name;}  
    public int getPrice() {return price;}  
    public int getNumberOfStock() {return numberOfStock;}  
    public int getSold() {return sold;}
```

```
}
```



상품 객체 배열 생성 및 사용 (2/2)

26

상품 (Goods) 객체 3개를 입력받아 배열에 저장하고, 그 값을 출력하시오.

```
import java.util.Scanner;
public class GoodsDemo {
    public static void main(String[] args){
        Goods [] goodsArray;
        goodsArray = new Goods [3];
        Scanner s = new Scanner(System.in);

        for(int i=0; i<goodsArray.length; i++) {
            String name = s.next();
            int price = s.nextInt();
            int n = s.nextInt();
            int sold = s.nextInt();
            goodsArray[i] = new Goods(name, price, n, sold);
        }
        for(int i=0; i<goodsArray.length; i++) {
            System.out.print(goodsArray[i].getName()+" ");
            System.out.print(goodsArray[i].getPrice()+" ");
            System.out.print(goodsArray[i].getNumberOfStock()+" ");
            System.out.println(goodsArray[i].getSold());
        }
    }
}
```

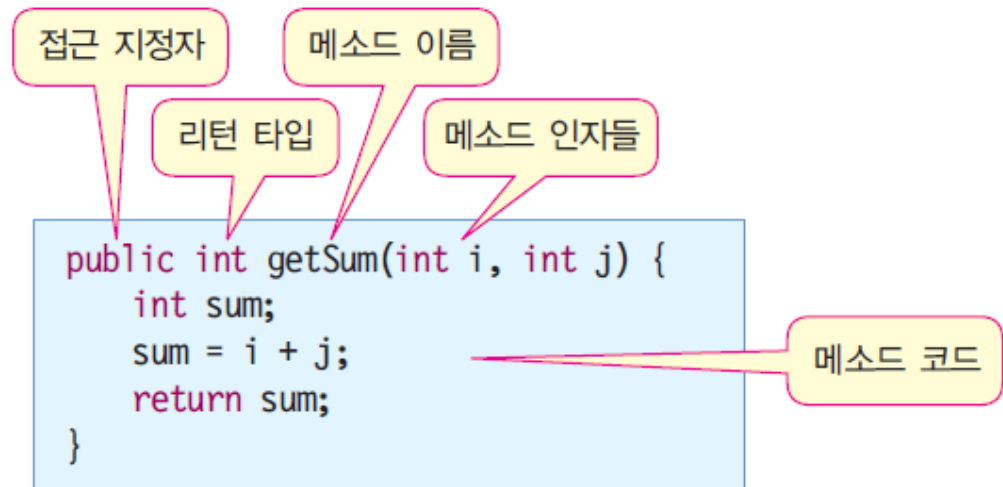
키 입력
부분

콜라 500 10 20
사이다 1000 20 30
맥주 2000 30 50
콜라 500 10 20
사이다 1000 20 30
맥주 2000 30 50

메소드 형식

27

- 메소드
 - ▣ 메소드는 함수이며, 함수 만드는 방법과 동일
 - ▣ 모든 메소드는 반드시 클래스 안에 있어야 함(캡슐화 원칙)
- 메소드 구성 형식
 - ▣ 접근 지정자
 - public, private, protected, default(접근 지정자 생략된 경우)
 - ▣ 리턴 타입
 - 메소드가 반환하는 결과값의 데이터 타입
 - ▣ 메소드 이름, 인자, 코드



인자 전달 - call by value

28

- 자바의 메소드 호출 시 인자 전달 방식
 - ▣ 값에 의한 호출(call by value)
- 기본 데이터 타입의 값을 전달하는 경우
 - ▣ 값이 복사되어 전달
 - ▣ 메소드의 매개 변수의 값이 변경되어도 호출한 인자의 값은 변경되지 않음
- 객체 혹은 배열을 전달하는 경우
 - ▣ 객체나 배열의 레퍼런스 만이 전달됨
 - 객체나 배열이 통째로 복사되어 전달되는 것이 아님
 - ▣ 메소드의 매개 변수와 호출한 인자가 서로 객체나 배열을 공유

call by value : 기본 데이터의 값 전달 사례

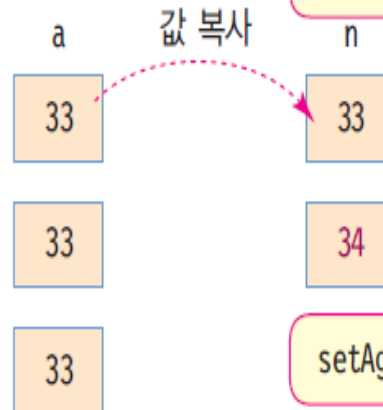
29

```
public class CallByValue {  
    public static void main (String args[]) {  
        Person aPerson = new Person("홍길동");  
        int a = 33;  
  
        aPerson.setAge(a);  
  
        System.out.println(a);  
    }  
}
```

aPerson.setAge(a);

System.out.println(a);

33



setAge()가 호출되면 매개변수 n이 생성된다.

```
public void setAge(int n) {  
    age = n;  
    n++;  
}
```

setAge()가 끝나면 n은 사라진다.

```
public class Person {  
    public String name;  
    public int age;  
  
    public Person(String s) {  
        name = s;  
    }  
}
```

```
public void setAge(int n) {  
    age = n;  
    n++;  
}
```

}

call by value : 객체 전달 사례

30

```
class MyInt {
    int val;
    MyInt(int i) {
        val = i;
    }
}

public class CallByValueObject {
    public static void main(String args[]) {
        Person aPerson = new Person("홍길동");
        MyInt a = new MyInt(33);

        aPerson.setAge(a);

        System.out.println(a.val);
    }
}
```

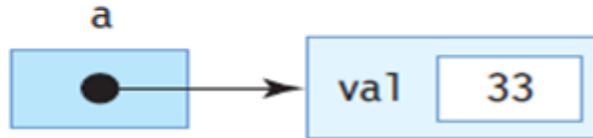
```
public class Person {
    public String name;
    public int age;
    public Person(String s) {
        name = s;
    }

    public void setAge(MyInt i) {
        age = i.val;
        i.val++;
    }
}
```

* 객체가 복사되어 전달되는 것이 아님
객체에 대한 레퍼런스 만이 복사되어 전달

```
MyInt a = new MyInt(33);
```

MyInt 객체 생성

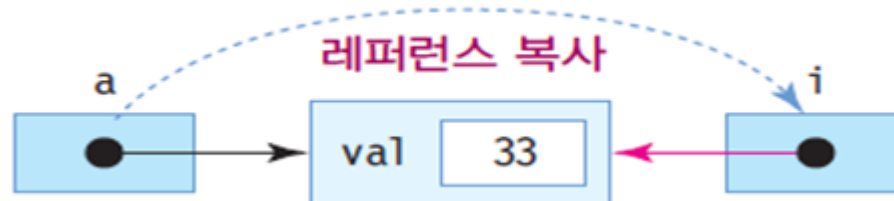


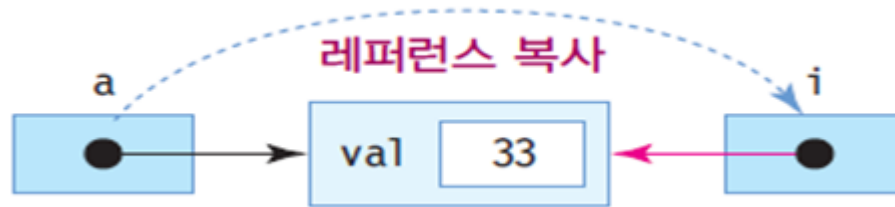
```
aPerson.setAge(a);
```

a값이 i에 전달됨

```
public void setAge(MyInt i)
```

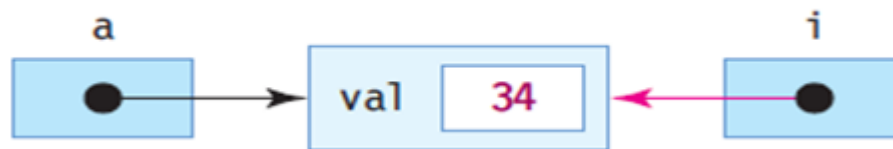
i와 a는 모두 동일한 객체를 가리킴





```
i.val++;
```

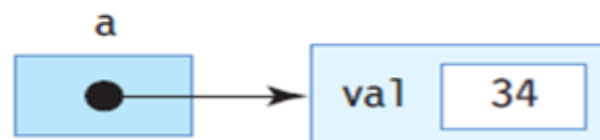
MyInt 객체의 val 값 1 증가



setAge() 메소드가 끝나면
레퍼런스 i가 사라짐

```
System.out.println(a.val);
```

34가 화면에 출력됨



call by value : 배열 전달 사례

33

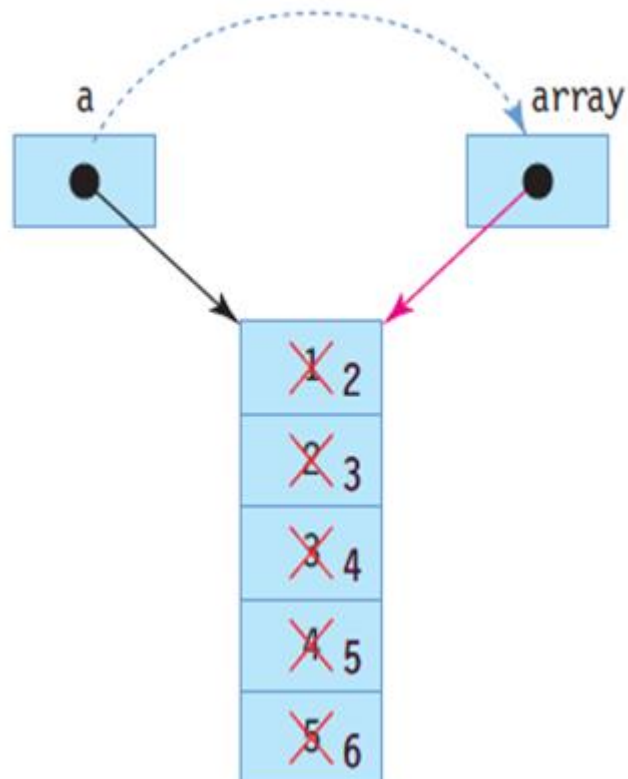
```
public class ArrayParameter {  
  
    public static void main(String args[]) {  
        int a[] = {1, 2, 3, 4, 5};  
  
        increase(a);  
  
        for(int i=0; i<a.length; i++)  
            System.out.print(a[i]+" ");  
    }  
}
```

2 3 4 5 6

- 인자로 배열을 전달하면
배열의 레퍼런스만이 전달됨

```
static void increase(int[] array) {  
    for(int i=0; i<array.length; i++) {  
        array[i]++;  
    }  
}
```

레퍼런스 복사



배열의 전달

34

char 배열을 메소드의 인자로 전달하여 배열 속의 공백(' ')문자를 ','로 대체하는 프로그램을 작성하시오.

```
public class ArrayParameter {
    static void replaceSpace(char a[]) {
        for (int i = 0; i < a.length; i++)
            if (a[i] == ' ')
                a[i] = ',';
    }
    static void printCharArray(char a[]) {
        for (int i = 0; i < a.length; i++)
            System.out.print(a[i]);
        System.out.println();
    }
}
```

This is a pencil.
This,is,a,pencil.

static

56쪽 참고

```
public class ArrayParameterDemo {
    public static void main (String args[]) {
        char c[] = {'T','h','i','s',' ','i','s',' ','a',
                    ' ','p','e','n','c','i','l','.'};
        ArrayParameter.printCharArray(c);
        ArrayParameter.replaceSpace(c);
        ArrayParameter.printCharArray(c);
    }
}
```

메소드 오버로딩 (overloading)

35

- 한 클래스 내에서 이름이 같은 두 개 이상의 메소드 작성
 - 메소드의 인자 개수 또는 타입이 달라야 한다.
 - 메소드의 리턴 타입만 달라서는 안된다. (컴파일 에러)

// 메소드 오버로딩이 성공한 사례

```
class MethodOverloading {  
    public int getSum(int i, int j) {  
        return i + j;  
    }  
    public int getSum(int i, int j, int k) {  
        return i + j + k;  
    }  
    public double getSum(double i, double j) {  
        return i + j;  
    }  
}
```

// 메소드 오버로딩이 실패한 사례

```
class MethodOverloadingFail {  
    public int getSum(int i, int j) {  
        return i + j;  
    }  
    public double getSum(int i, int j) {  
        return (double)(i + j);  
    }  
}
```

오버로딩된 메소드 호출

36

```
public static void main(String args[]) {  
    MethodSample a = new MethodSample();  
  
    int i = a.getSum(1, 2);  
  
    int j = a.getSum(1, 2, 3);  
  
    double k = a.getSum(1.1, 2.2);  
}
```

```
public class MethodSample {  
    ➤ public int getSum(int i, int j) {  
        return i + j;  
    }  
  
    ➤ public int getSum(int i, int j, int k) {  
        return i + j + k;  
    }  
  
    ➤ public double getSum(double i, double j) {  
        return i + j;  
    }  
}
```

this 레퍼런스

37

- this의 기초 개념
 - ▣ 현재 객체 자기 자신을 가리킴
 - 자기 자신에 대한 레퍼런스
 - 같은 클래스 내에서 클래스 멤버, 변수를 접근할 때 객체 이름이 없으면 묵시적으로 this로 가정
- this의 필요성
 - ▣ 객체의 멤버 변수와 메소드 변수의 이름이 같은 경우
 - ▣ 객체 자신을 메소드에 전달 또는 반환할 때

this에 대한 이해

38

```
public class Samp {  
    int id;  
    public Samp(int x){  
        this.id = x;  
    }  
    public void set(int x){  
        this.id = x;  
    }  
    public int get(){  
        return this.id;  
    }  
}
```

```
public class SampDemo {  
    public static void main(String[] args) {  
        Samp ob1 = new Samp(3);  
        Samp ob2 = new Samp(3);  
        Samp ob3 = new Samp(3);  
  
        ob1.set(5);  
        ob2.set(6);  
        ob3.set(7);  
  
        System.out.println(ob1.get());  
        System.out.println(ob2.get());  
        System.out.println(ob3.get());  
    }  
}
```

this : 객체의 멤버 변수와 메소드 변수의 이름이 같은 경우

39

```
public class Samp {  
    int id;  
    public Samp(int id){  
        this.id = id;  
    }  
    public void set(int id){  
        this.id = id;  
    }  
    public int get(){  
        return this.id;  
    }  
}
```

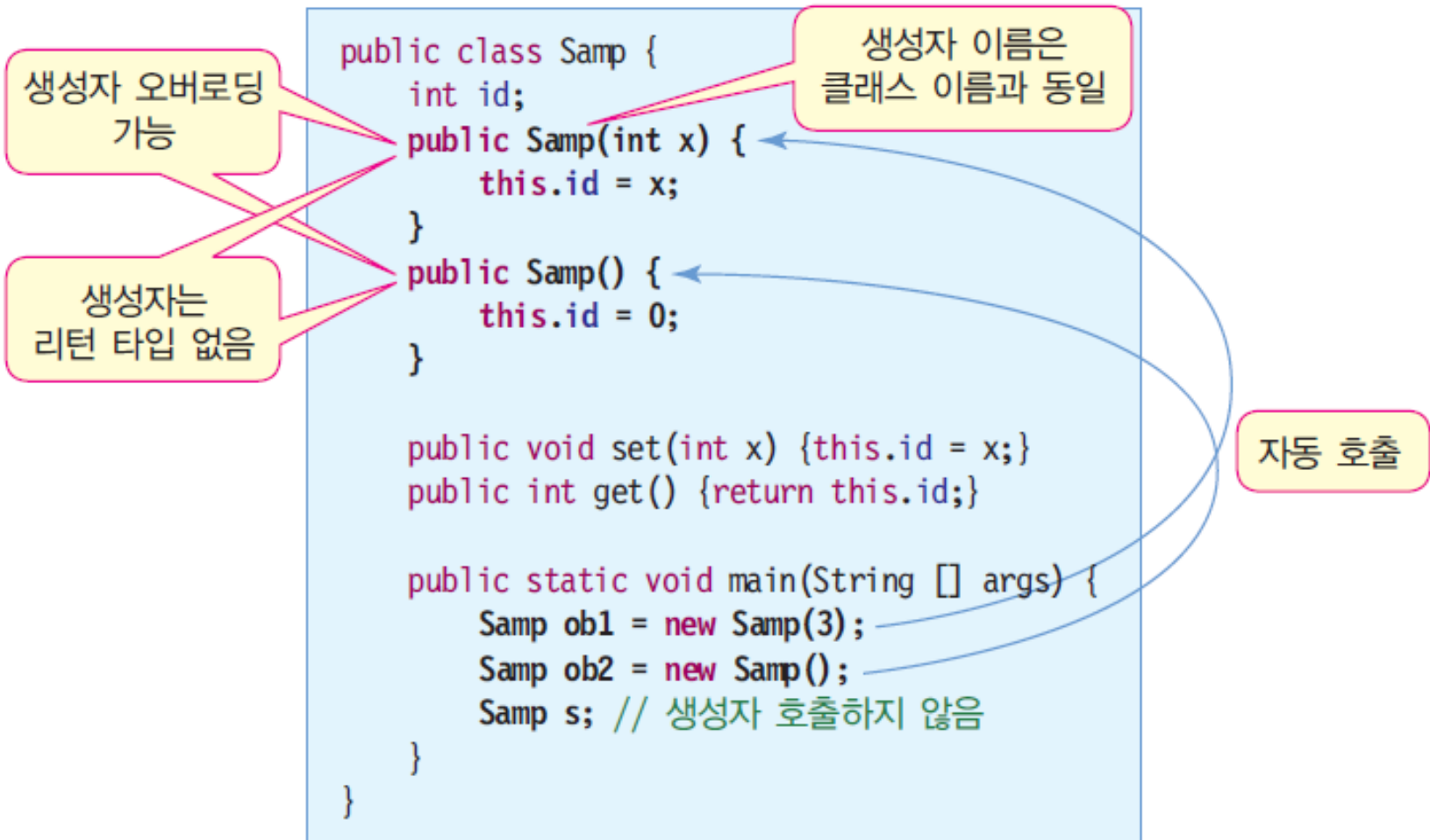
생성자 (constructor)

40

- ▣ 생성자는 메소드이다.
- ▣ 생성자의 이름은 클래스 이름과 동일하다.
- ▣ 생성자는 new를 통해 객체를 생성할 때만 호출된다.
- ▣ 생성자도 오버로딩이 가능하다.
- ▣ 생성자는 리턴 타입을 지정할 수 없다.
- ▣ 생성자는 하나 이상 정의되어야 한다.
 - 개발자가 생성자를 하나도 정의하지 않으면 자동으로 기본 생성자가 정의된다.
 - 기본 생성자는 디폴트 생성자 (default constructor) 라고도 하며, 컴파일러에 의해 자동으로 생성된다.

생성자 정의와 생성자 호출

41



생성자 정의와 호출

42

클래스 Book을 String title, String author, int ISBN의 3개의 필드를 갖도록 정의하라.

```
public class Book {  
    String title;  
    String author;  
    int ISBN;  
    public Book(String title, String author, int ISBN) {  
        this.title = title;  
        this.author = author;  
        this.ISBN = ISBN;  
    }  
}
```

정의

```
Book OOPBook = new Book("Java", "Kim", 1234);
```

호출

기본 생성자, 디폴트 생성자

43

```
public class DefaultConstructor {  
    int x;  
    public void setX(int x) {this.x = x;}  
    public int getX() {return x;}  
}  
public class DefaultConstructorDemo1 {  
    public static void main(String [] args) {  
        DefaultConstructor p= new DefaultConstructor();  
        p.setX(3);  
        System.out.println(p.getX());  
    }  
}
```

개발자가 작성한 코드

호출

```
public class DefaultConstructor {  
    int x;  
    public DefaultConstructor() { }  
    public void setX(int x) {this.x = x;}  
    public int getX() {return x;}  
}
```

컴파일러에 의해
자동 삽입된 기본 생성자

기본 생성자가 자동 생성되지 않는 경우

44

- 클래스에 생성자가 하나라도 존재하면 자동으로 기본 생성자가 생성되지 않음

컴파일러가 기본 생성자를 자동 생성하지 않음

public DefaultConstructor() { }

```
public class DefaultConstructor {  
    int x;  
    public void setX(int x) {this.x = x;}  
    public int getX() {return x;}  
  
    public DefaultConstructor(int x) {  
        this.x = x;  
    }  
}  
  
public class DefaultConstructorDemo2 {  
    public static void main(String [] args) {  
        DefaultConstructor p1= new DefaultConstructor(3);  
        int n = p1.getX();  
        DefaultConstructor p2= new DefaultConstructor();  
        p2.setX(5);  
    }  
}
```

컴파일 오류.
해당하는 생성자가
없음 !!!

this(), 생성자에서 다른 생성자 호출

45

□ this()

- 같은 클래스의 다른 생성자 호출 시 사용
- 생성자 내에서만 사용 가능 (다른 메소드에서는 사용 불가)
- 반드시 생성자 코드의 제일 처음에 수행

```
public class Book {
    String title;
    String author;
    int ISBN;

    public Book(String title, String author, int ISBN) {
        this.title = title;
        this.author = author;
        this.ISBN = ISBN;
    }

    public Book(String title, int ISBN) {
        this(title, "Anonymous", ISBN);
    }

    public Book() {
        this(null, null, 0);
        System.out.println("생성자가 호출되었음");
    }
}

public class BookDemo2 {
    public static void main(String [] args) {
        Book OOPBook = new Book("Java", "Kim", 1234);
        Book holyBible = new Book("Holy Bible", 1);
        Book emptyBook = new Book();
    }
}
```

title = "Holy Bible"
author = 'Anonymous'
ISBN = 1

title = "Holy Bible"
ISBN = 1

this() 사용 실패 예

46

```
public Book() {  
    System.out.println("생성자가 호출되었음");  
    this(null, null, 0); // 생성자의 첫 번째 문장이 아니기 때문에 컴파일 오류  
}
```

객체의 소멸과 Garbage Collection

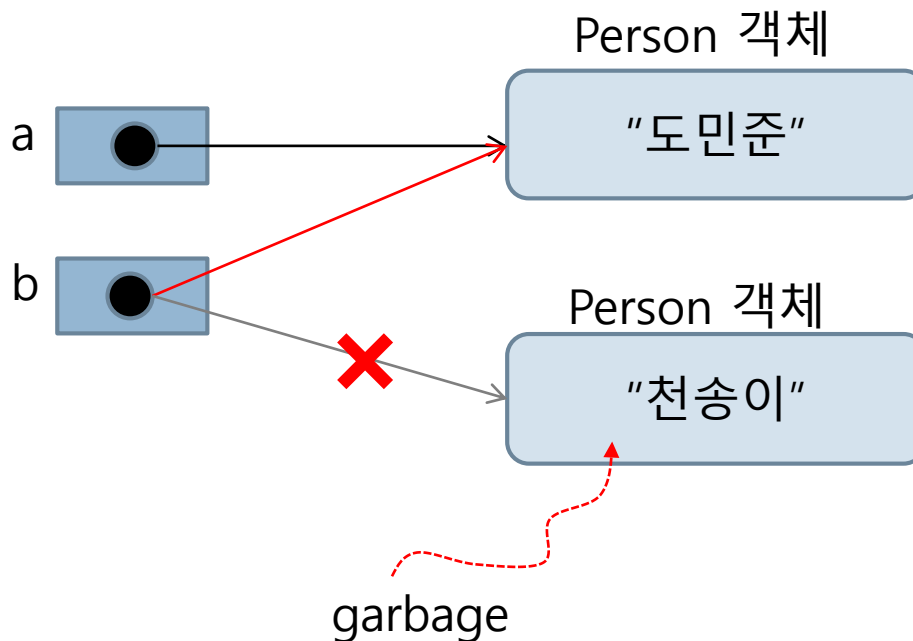
47

- 객체 소멸
 - ▣ new에 의해 생성된 객체에 할당되었던 메모리를 JVM에게 되돌려 주는 것
 - ▣ 가용 메모리에 포함시킴
- 자바는 객체 삭제 기능 없음
 - ▣ 개발자에게는 매우 다행스러운 것
 - C/C++에서는 할당받은 객체를 개발자가 프로그램 내에서 삭제해야 함
- Garbage Collection
 - ▣ 객체에 대한 레퍼런스가 없어지면 객체는 가비지(garbage)가 됨
 - ▣ JVM 의 garbage collector 가 garbage 를 반환

(예제) Garbage 가 생기는 경우

48

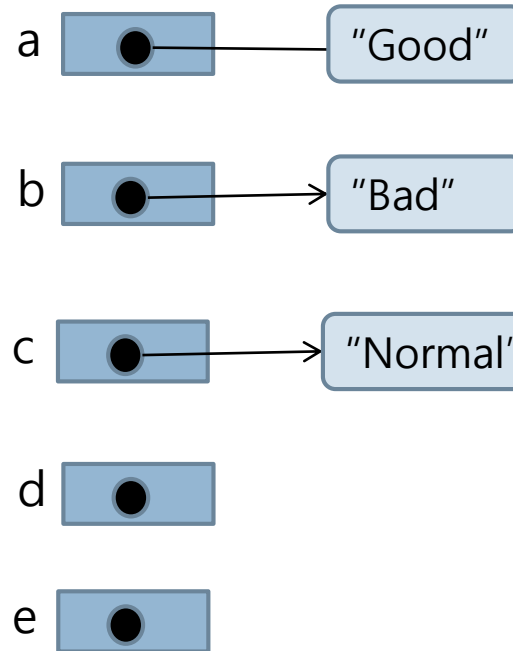
```
public class PersonDemo1 {  
    public static void main(String[] args) {  
        Person a, b;  
        a = new Person("도민준");  
        b = new Person("천송이");  
        b = a;    // b가 가리키던 객체는 garbage 가 된다.  
    }  
}
```



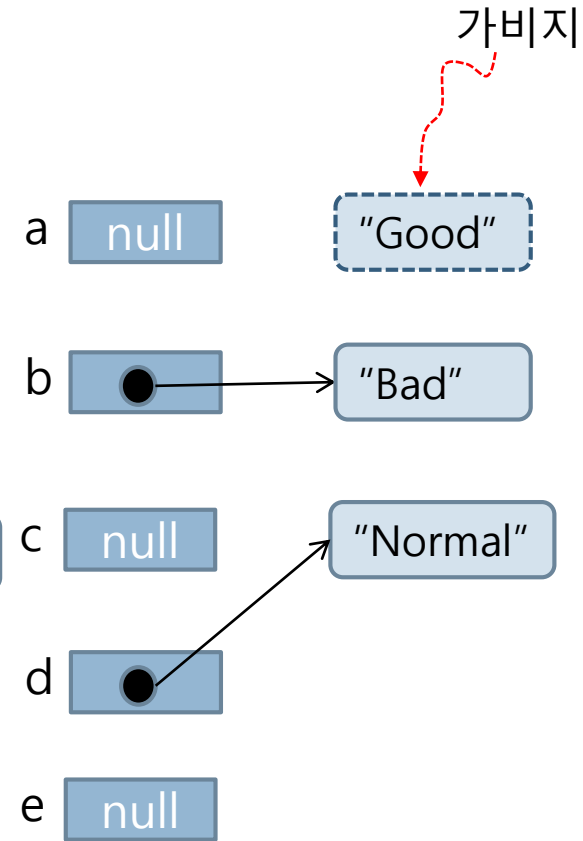
Garbage 발생

49

```
public class GarbageDemo1 {  
    public static void main(String[] args) {  
        String a = new String("Good");  
        String b = new String("Bad");  
        String c = new String("Normal");  
        String d, e;  
        a = null;  
        d = c;  
        c = null;  
    }  
}
```



(A) 초기 객체 생성시



(B) 코드 전체 실행 후

Garbage Collection

50

- 자동 실행
 - ▣ JVM 에 포함된 가비지 컬렉터(garbage collector)가 자동으로 실행

- 개발자가 강제로 실행하려는 경우
 - ▣ System 또는 Runtime 객체의 gc() 메소드를 호출한다.

```
System.gc(); // 가비지 컬렉션 작동 요청
```
 - ▣ 그러나~ garbage collection 시점은 JVM 이 전적으로 판단한다.

클래스 접근 지정자

51

- 클래스 앞에 올 수 있는 접근 지정자
 - ▣ public 접근 지정자

```
public class Person { }
```

- 다른 모든 클래스가 접근 가능

- ▣ 접근 지정자 생략 (default 접근 지정자)

```
class Person { }
```

- 또는 package-private라고도 함
- 같은 패키지 내에 있는 클래스에서만 접근 가능
 - 즉, 동일한 폴더 속에 있는 클래스끼리 접근 가능

멤버 접근 지정자

52

default (또는 package-private)	<ul style="list-style-type: none">같은 패키지 내에서 접근 가능
public	<ul style="list-style-type: none">패키지 내부, 외부 클래스에서 접근 가능
private	<ul style="list-style-type: none">정의된 클래스 내에서만 접근 가능상속 받은 하위 클래스에서도 접근 불가
protected	<ul style="list-style-type: none">같은 패키지 내에서 접근 가능다른 패키지에서 접근은 불가하나 상속을 받은 경우 하위 클래스에서는 접근 가능

멤버에 접근하는 클래스	멤버의 접근 지정자			
	default	private	protected	public
같은 패키지의 클래스	O	X	O	O
다른 패키지의 클래스	X	X	X	O

default 접근 지정자 사례

53

```
class A {  
    void f() {  
        B b = new B();  
        b.n = 3;  
        b.g();  
    }  
}
```

패키지 P

```
public class B {  
    int n;  
    void g() {  
        n = 5;  
    }  
}
```

```
class C {  
    public void k() {  
        B b = new B();  
        b.n = 7;  
        b.g();  
    }  
}
```

protected 접근 지정자 사례

```
class A {  
    void f() {  
        B b = new B();  
        b.n = 3;  
        b.g();  
    }  
}
```

패키지 P

```
public class B {  
    protected int n;  
    protected void g() {  
        n = 5;  
    }  
}
```

```
class C {  
    public void k() {  
        B b = new B();  
        b.n = 7;  
        b.g();  
    }  
}
```

D가 B를 상속받음

```
class D extends B {  
    void f() {  
        n = 3;  
        g();  
    }  
}
```

접근 지정자의 사용 (1/2)

54

다음의 소스를 컴파일 해보고 오류가 난 이유를 설명하고 오류를 수정하십시오.

```
public class Sample {
    public int a;
    private int b;
    int c;
}

public class AccessDemo1 {
    public static void main(String[] args) {
        Sample aClass = new Sample();
        aClass.a = 10;
        aClass.b = 10;
        aClass.c = 10;
    }
}
```

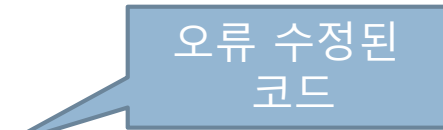
접근 지정자의 사용 (2/2)

55

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
The field Sample.b is not visible
```

```
at AccessEx.main(AccessEx.java:11)
```

```
class Sample {  
    public int a;  
    private int b;  
    int c;  
    public int getB() {  
        return b;  
    }  
    public void setB(int value) {  
        b = value;  
    }  
}  
public class AccessDemo1 {  
    public static void main(String[] args) {  
        Sample aClass = new Sample();  
        aClass.a = 10;  
        aClass.setB(10);  
        aClass.c = 10;  
    }  
}
```



- Sample 클래스의 a와 c는 각각 public, default 지정자로 선언이 되었으므로 같은 패키지에 속한 AccessEx 클래스에서 접근이 가능
- b는 private으로 선언이 되었으므로 AccessEx 클래스에서는 접근이 불가능
- private 접근 지정자를 갖는 멤버는 클래스 내부에 get/set 메소드를 만들어서 접근한다.

static 멤버와 non-static 멤버

56

- non-static 멤버 (인스턴스 멤버)
 - ▣ 객체마다 독립적으로 존재한다.
 - ▣ 각 필드와 메소드는 객체가 생성된 후에 사용할 수 있다.
- static 멤버 (클래스 멤버)
 - ▣ 클래스에 하나만 생성된다.
 - ▣ 클래스가 로딩될 때 공간을 할당 받는다.
 - ▣ 동일한 클래스의 모든 객체에 의해 공유된다.

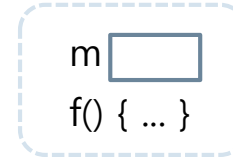
```
public class StaticSample {  
    int n;                                // non-static 필드  
    void g() { }                          //non-static 메소드  
    static int m;                         // static 필드  
    static void f() { }                  // static 메소드  
}
```

See also p. 61

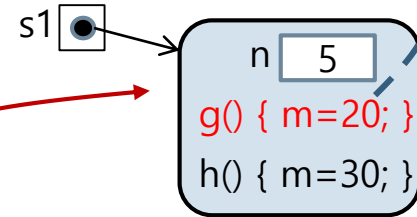
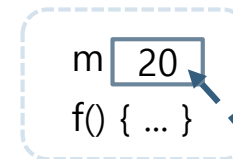
static 멤버를 객체의 멤버로 접근하는 사례 (1/2)

57

```
public class StaticSample {  
    public int n;  
    public void g() {  
        m = 20;  
    }  
    public void h() {  
        m = 30;  
    }  
    public static int m;  
    public static void f() {  
        m = 5;  
    }  
}
```

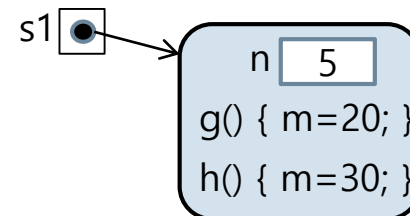
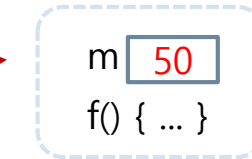


static 멤버 생성



`s1.g()`호출에 의해
static 멤버 `m`의
값이 20으로 설정

```
public class StaticDemo1 {  
    public static void main(String[] args) {  
        StaticSample s1, s2;  
        s1 = new StaticSample();  
        s1.n = 5;  
        s1.g();  
        s1.m = 50; // static  
        s2 = new StaticSample();  
        s2.n = 8;  
        s2.h();  
        s2.f(); // static  
        System.out.println(s1.m);  
    }  
}
```



`s1.m=50;`에 의해
static 멤버 `m`의
값이 50으로 설정

실행결과

5

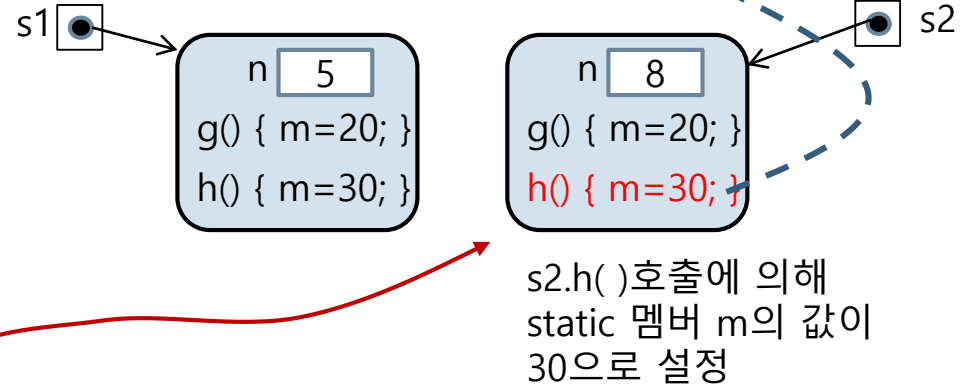
static 멤버를 객체의 멤버로 접근하는 사례 (2/2)

58

s1, s2에 의해 공유

m 30
f() { ... }

```
public class StaticSample {  
    public int n;  
    public void g() {  
        m = 20;  
    }  
    public void h() {  
        m = 30;  
    }  
    public static int m;  
    public static void f() {  
        m = 5;  
    }  
}
```

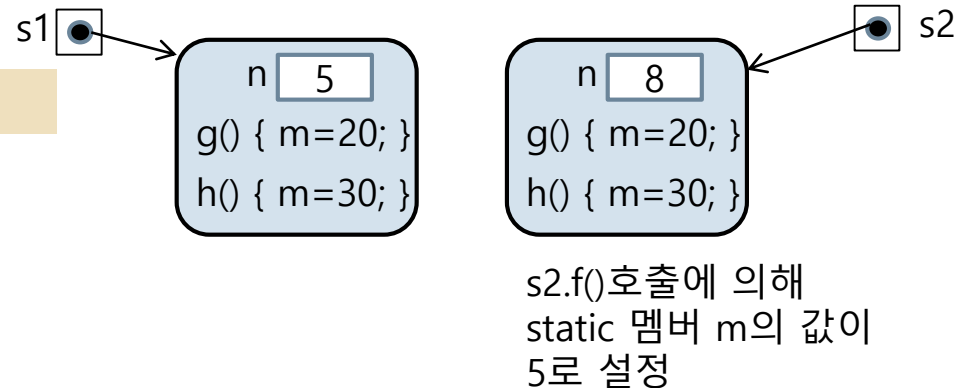


```
public class StaticDemo1 {  
    public static void main(String[] args) {  
        StaticSample s1, s2;  
        s1 = new StaticSample();  
        s1.n = 5;  
        s1.g();  
        s1.m = 50; // static  
        s2 = new StaticSample();  
        s2.n = 8;  
        s2.h();  
        s2.f(); // static  
        System.out.println(s1.m);  
    }  
}
```

s1, s2에 의해 공유

m 5
f() { m=5; }

s2.f();



System.out.println(s1.m);

5 출력

static 멤버를 클래스 이름으로 접근하는 사례

59

```
public class StaticSample {  
    public int n;  
    public void g() {  
        m = 20;  
    }  
    public void h() {  
        m = 30;  
    }  
    public static int m;  
    public static void f() {  
        m = 5;  
    }  
}
```

```
public class StaticDemo1 {  
    public static void main(String[] args) {  
        StaticSample.m = 10;  
  
        StaticSample s1;  
        s1 = new StaticSample();  
        System.out.println(s1.m);  
        s1.f();  
        StaticSample.f();  
    }  
}
```

```
StaticSample.m = 10;
```

```
m 10
f() { ... }
```

static 멤버 생성

```
StaticSample s1;
s1 = new StaticSample( );
```

s1

```
m 10
f() { ... }
```

```
n
g() { m=20; }
h() { m=30; }
```

객체 s1 생성

```
System.out.println(s1.m);
```

10 출력

```
s1.f( );
```

```
m 5
f() { ... }
```

s1

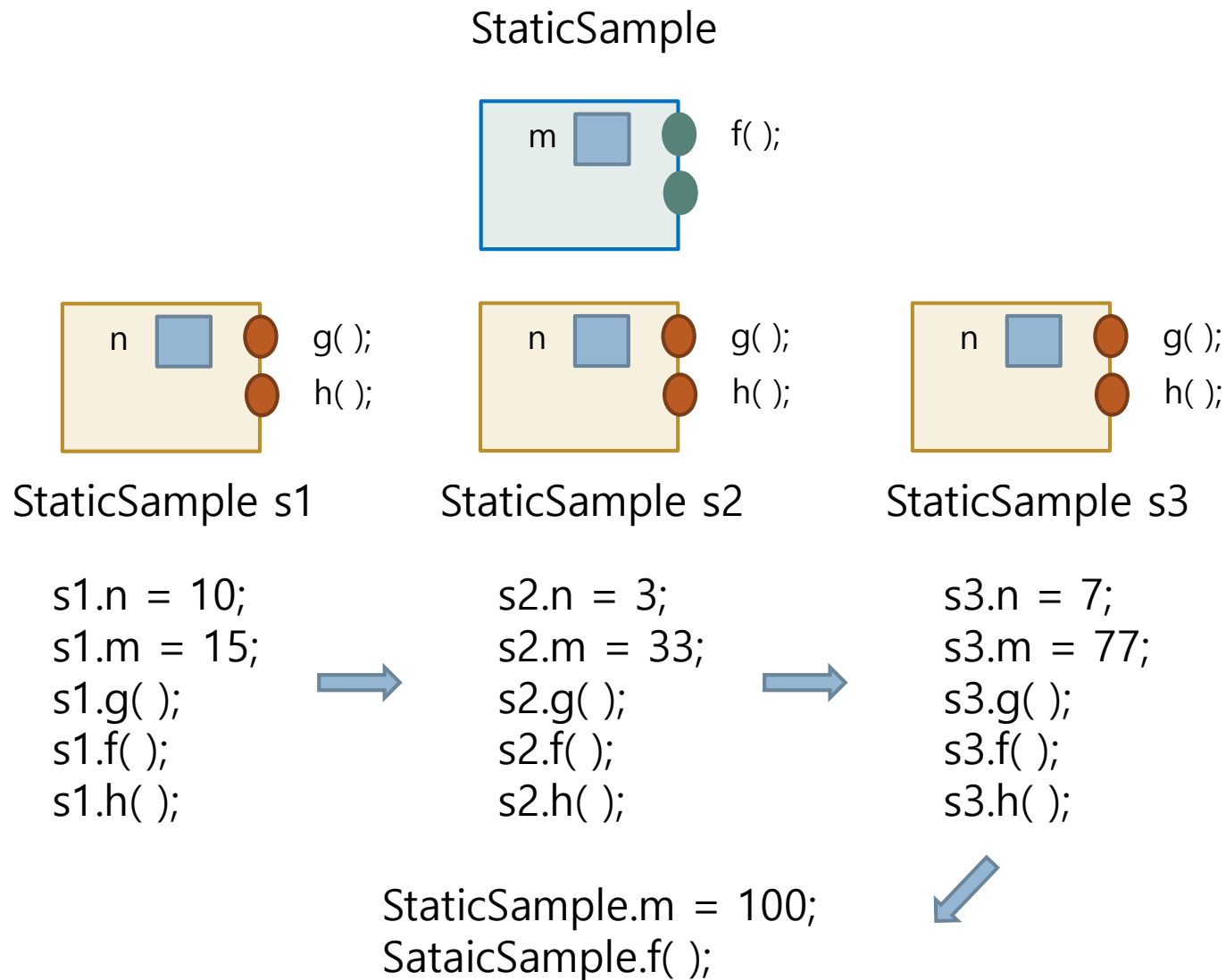
```
n
g() { m=20; }
h() { m=30; }
```

```
StaticSample.f( );
```

s1.f();의 호출과 동일함

Static 에 대한 이해 – 추가 자료

61



static의 활용

62

- 전역 변수와 전역 함수를 만들 때 활용
 - ▣ 자바에서의 캡슐화 원칙
 - 다른 모든 클래스에서 공유하는 전역 변수나 전역 함수도 클래스 내부에만 정의
- java.lang.Math 클래스
 - ▣ JDK와 함께 배포되는 java.lang.Math 클래스
 - ▣ 모든 메소드가 static으로 정의되어 다른 모든 클래스에서 사용됨
 - ▣ 객체를 생성하지 않고 바로 호출할 수 있는 상수와 메소드 제공

```
public class Math {  
    static int abs(int a);  
    static double cos(double a);  
    static int max(int a, int b);  
    static double random();  
    ...  
}
```

// 권하지 않는 사용법

```
Math m = new Math();  
int n = m.abs(-5);
```

// 바른 사용법

```
int n = Math.abs(-5);
```

static 메소드의 제약 조건

63

- static 메소드는 오직 static 멤버만 접근 가능
 - ▣ 객체가 생성되지 않은 상황에서도 사용이 가능하므로 객체에 속한 인스턴스 메소드, 인스턴스 변수 등 사용 불가
 - ▣ 인스턴스 메소드는 static 멤버들을 모두 사용 가능

- static 메소드에서는 this 키워드를 사용할 수 없음
 - ▣ 객체가 생성되지 않은 상황에서도 호출이 가능하기 때문에 현재 실행 중인 객체를 가리키는 this 레퍼런스를 사용할 수 없음

static을 이용한 달러와 원화 사이의 변환 예제

64

```
public class CurrencyConverter {  
    private static double rate;    // 원화에 대한 환율  
    public static double toDollar(double won) {  
        return won/rate;          // 원화를 달러로 변환  
    }  
    public static double toKWR(double dollar) {  
        return dollar * rate;     // 달러를 원화로 변환  
    }  
    public static void setRate(double r) {  
        rate = r;                 // 환율 설정. KWR/$1  
    }  
}
```

실행 결과

백만원은 892.0606601248885달러입니다.
백달러는 112100.0원입니다.

```
public class CurrencyConverterDemo {  
    public static void main(String[] args) {  
        CurrencyConverter.setRate(1121);    // 달러 환율 설정  
        System.out.println("백만원은 " + CurrencyConverter.toDollar(1000000)  
            + "달러입니다.");  
        System.out.println("백달러는 " + CurrencyConverter.toKWR(100)  
            + "원입니다.");  
    }  
}
```


final

65

- final 클래스 - 더 이상 클래스 상속 불가능

```
final class FinalClass {
    ....
}
class DerivedClass extends FinalClass {    // 컴파일 오류 발생
    ....
}
```

- final 메소드 - 더 이상 오버라이딩 불가능

```
public class SuperClass {
    protected final int finalMethod() { ... }
}

class DerivedClass extends SuperClass {
    protected int finalMethod() { ... }
    // 컴파일 오류, 오버라이딩 할 수 없음
}
```

final 필드

66

- final 필드, 상수 정의
 - ▣ 선언시에 초기값을 지정하며 값을 변경할 수 없다.

```
class SharedClass {  
    public static final double PI = 3.141592653589793;  
}
```

```
public class FinalFieldClass {  
    final int ROWS = 10; // 상수 정의, 이때 초기 값(10)을 반드시 설정  
    final int COLS; // 컴파일 오류, 초기값을 지정하지 않았음  
    void f() {  
        int [] intArray = new int [ROWS]; // 상수 활용  
        ROWS = 30; // 컴파일 오류 발생, final 필드 값을 변경할 수 없다.  
    }  
}
```