제 4 장 클래스와 객체

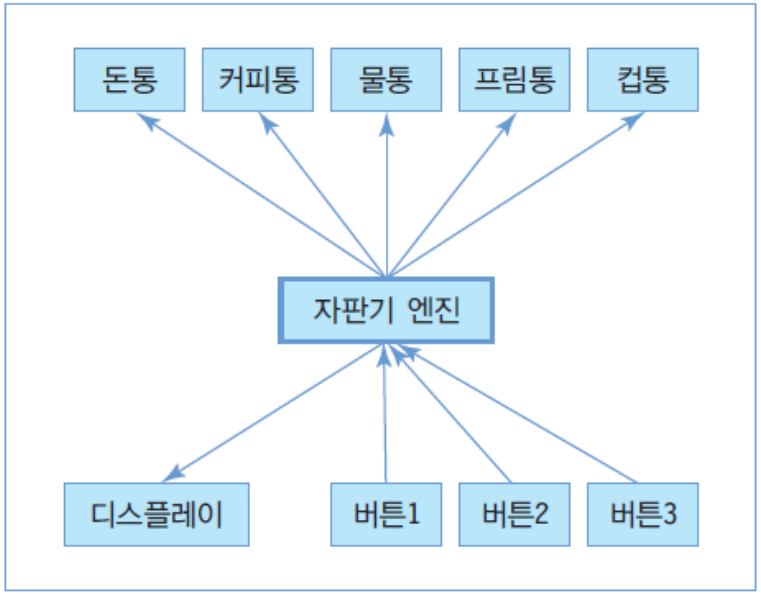
객체지향 언어의 목적

- 소프트웨어의 생산성 향상
  - 컴퓨터 산업 발전에 따라 소프트웨어의 생명 주기(life cycle) 단축
  - 객체 지향 언어는 상속, 다형성, 객체, 캡슐화 등 소프트웨어 재사용을 위한 여러 장치 내장
    - 소프트웨어의 재사용과 부분 수정을 통해 소프트웨어를 다시 만드는 부담을 대폭 줄임으로써 소프트웨어의 생산성이 향상
- 실세계에 대한 쉬운 모델링
  - 과거
    - 수학 계산/통계 처리를 하는 등의 처리 과정, 계산 절차가 중요
  - 현재
    - 컴퓨터가 산업 전반에 활용
    - 실세계에서 발생하는 일을 프로그래밍
      - 실세계에서는 절차나 과정보다 일과 관련된 물체(객체)등의 상호 작용으로 묘사하는 것이 용이
  - 실세계의 일을 보다 쉽게 프로그래밍하기 위한 객체 중심의 객체 지향 언어 탄생

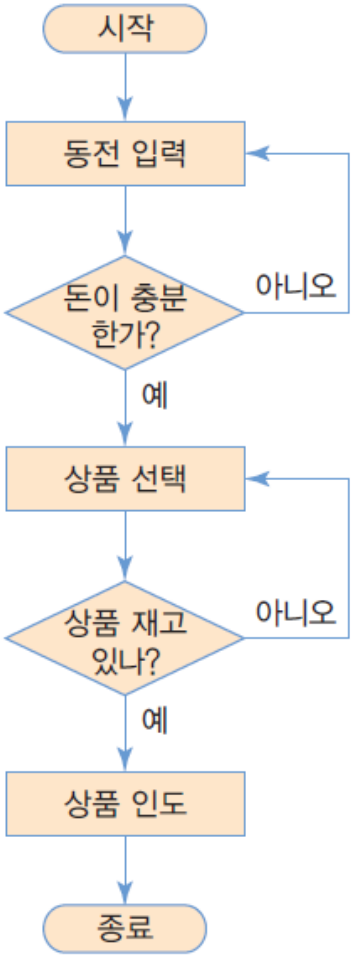
절차 지향 프로그래밍과 객체 지향 프로그래밍

- 절차 지향 프로그래밍
  - 작업 순서를 표현하는 컴퓨터 명령 집합
  - 함수들의 집합으로 프로그램 작성
- 객체 지향 프로그래밍
  - 프로그램을 실제 세상에 가깝게 모델링
  - 컴퓨터가 수행하는 작업을 객체들간의 상호 작용으로 표현
  - 클래스 혹은 객체들의 집합으로 프로그램 작성

커피 자판기



객체지향적 프로그래밍의 객체들의 상호 관련성

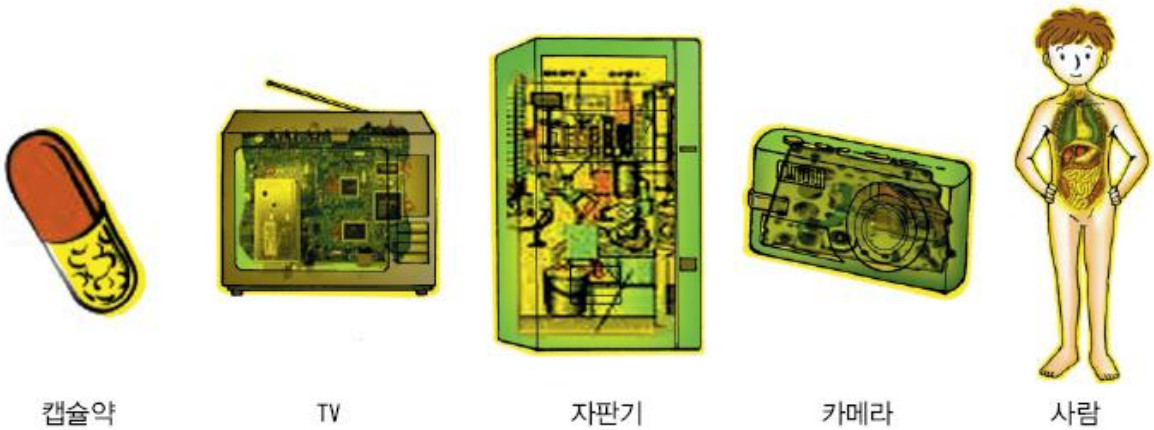


절차지향적 프로그래밍의 실행 절차

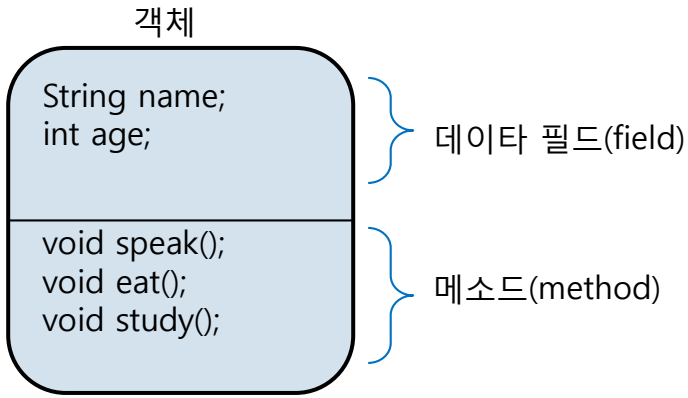
객체 지향 언어의 특성 : 캡슐화

- 캡슐화
  - 메소드(함수)와 데이터를 클래스 내에 선언하고 구현
  - 외부에서는 공개된 메소드의 인터페이스만 접근 가능
    - 외부에서는 비공개 데이터에 직접 접근하거나 메소드의 구현 세부를 알 수 없음
  - 객체 내 데이터에 대한 보안, 보호, 외부 접근 제한

실세계의 캡슐화

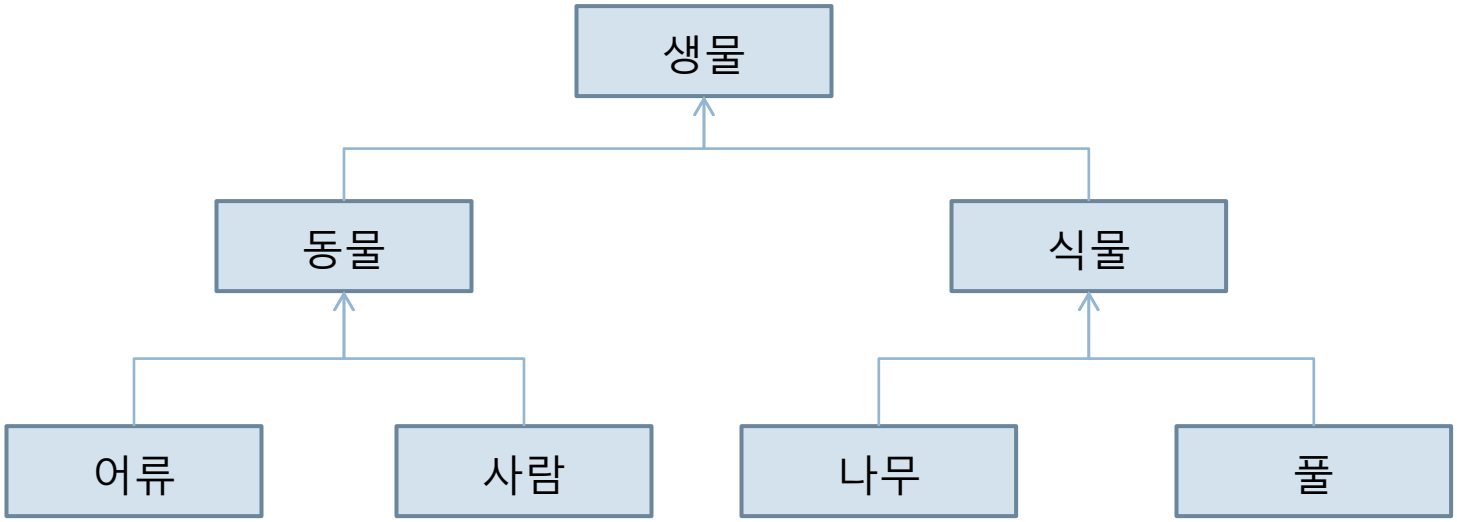


자바 객체의 캡슐화



객체 지향의 특성 : 상속

실세계에서의 상속 예 - 유전적인 상속 관계 표현



객체 지향 언어에서의 상속


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

상속

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

Animal 객체

```
String name;
int age;

void eat();
void sleep();
void love();
```

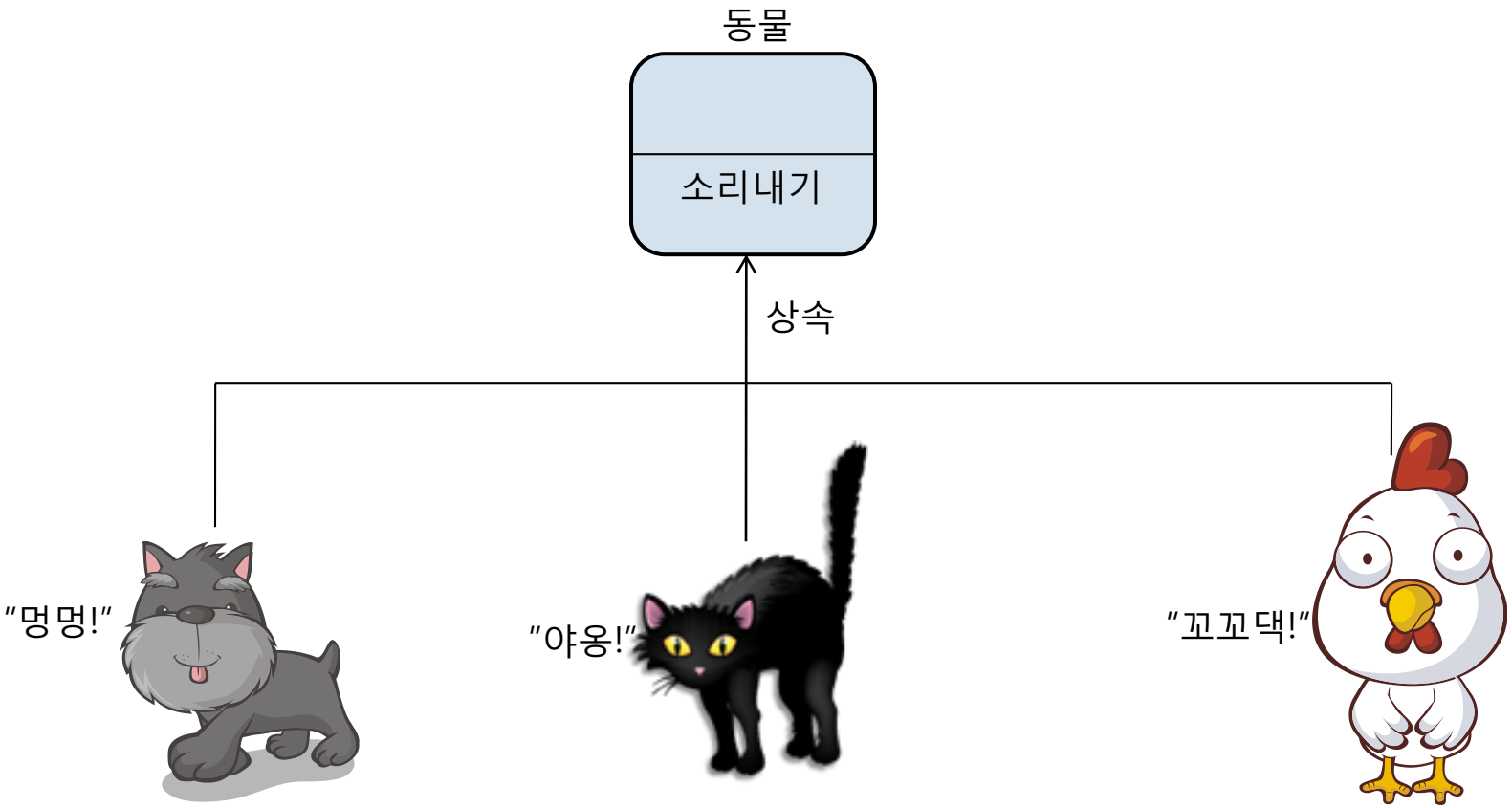
Human 객체

```
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 : 수퍼 클래스, 하위 클래스 : 서브 클래스
  - 서브 클래스
    - 수퍼 클래스 코드의 재사용
    - 새로운 특성 추가 가능
  - 자바에 다중 상속 없음
    - 인터페이스를 통해 다중 상속과 같은 효과 얻음

객체 지향의 특성 : 다형성

- 다형성
  - 같은 메시지 또는 함수에서 객체에 따라서 다른 동작 가능함
  - 다형성은 오버라이딩과 밀접한 관계가 있음

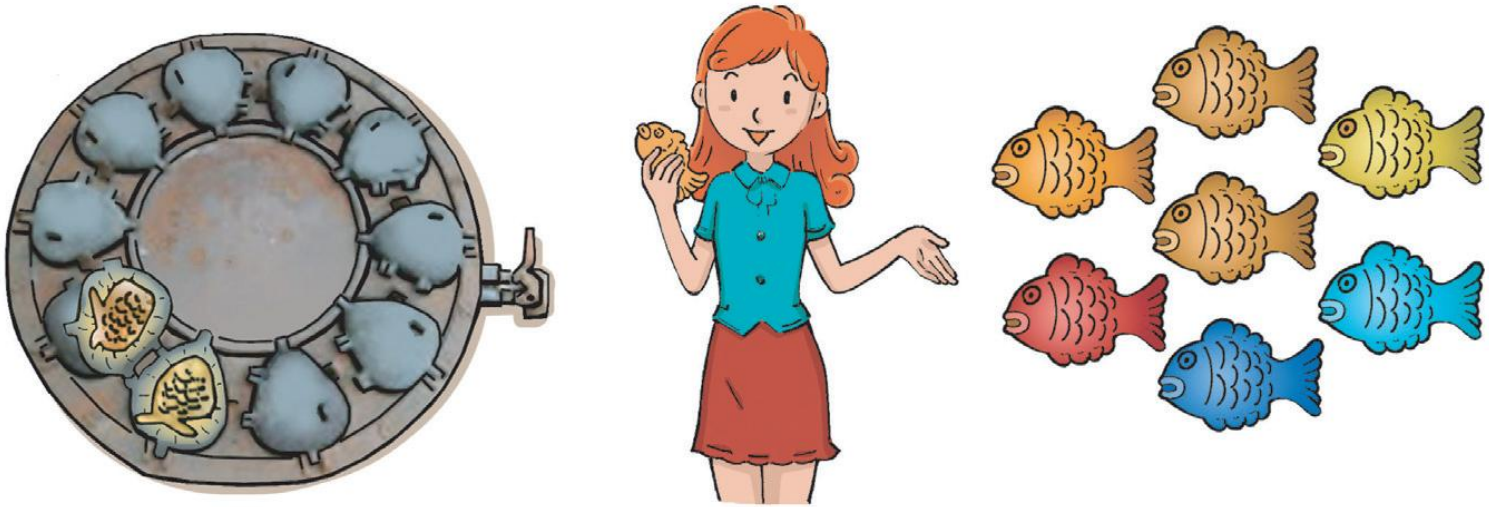


클래스와 객체

- 클래스
  - 객체의 공통된 특징 기술
  - 객체의 특성과 행위 선언
- 객체
  - 물리적 공간을 갖는 구체적인 실체
  - 클래스의 인스턴스(실체)
    - 클래스를 구체화한 객체를 인스턴스(instance)라고 부름
    - 객체와 인스턴스는 같은 뜻으로 사용
- 사례
  - 클래스: 소나타자동차, 객체: 출고된 실제 소나타 100대
  - 클래스: 벽시계, 객체: 우리집 벽에 걸린 벽시계들
  - 클래스: 책상, 객체: 우리가 사용중인 실제 책상들

클래스와 객체와의 관계

붕어빵 틀은 클래스이며, 이 틀의 형태로 구워진 붕어빵은 바로 객체입니다. 붕어빵은 틀의 모양대로 만들어지지만 서로 조금씩 다릅니다.
치즈붕어빵, 크림붕어빵, 앙코붕어빵 등이 있습니다.
그래도 이들은 모두 붕어빵입니다.



사람을 사례로 든 클래스와 객체 사례

클래스: 사람

이름, 직업,나이, 성별, 혈액형
밥 먹기, 잠자기, 말하기, 걷기

이름 최승희
직업 의사
나이 45
성별 여
혈액형 A

객체 : 최승희

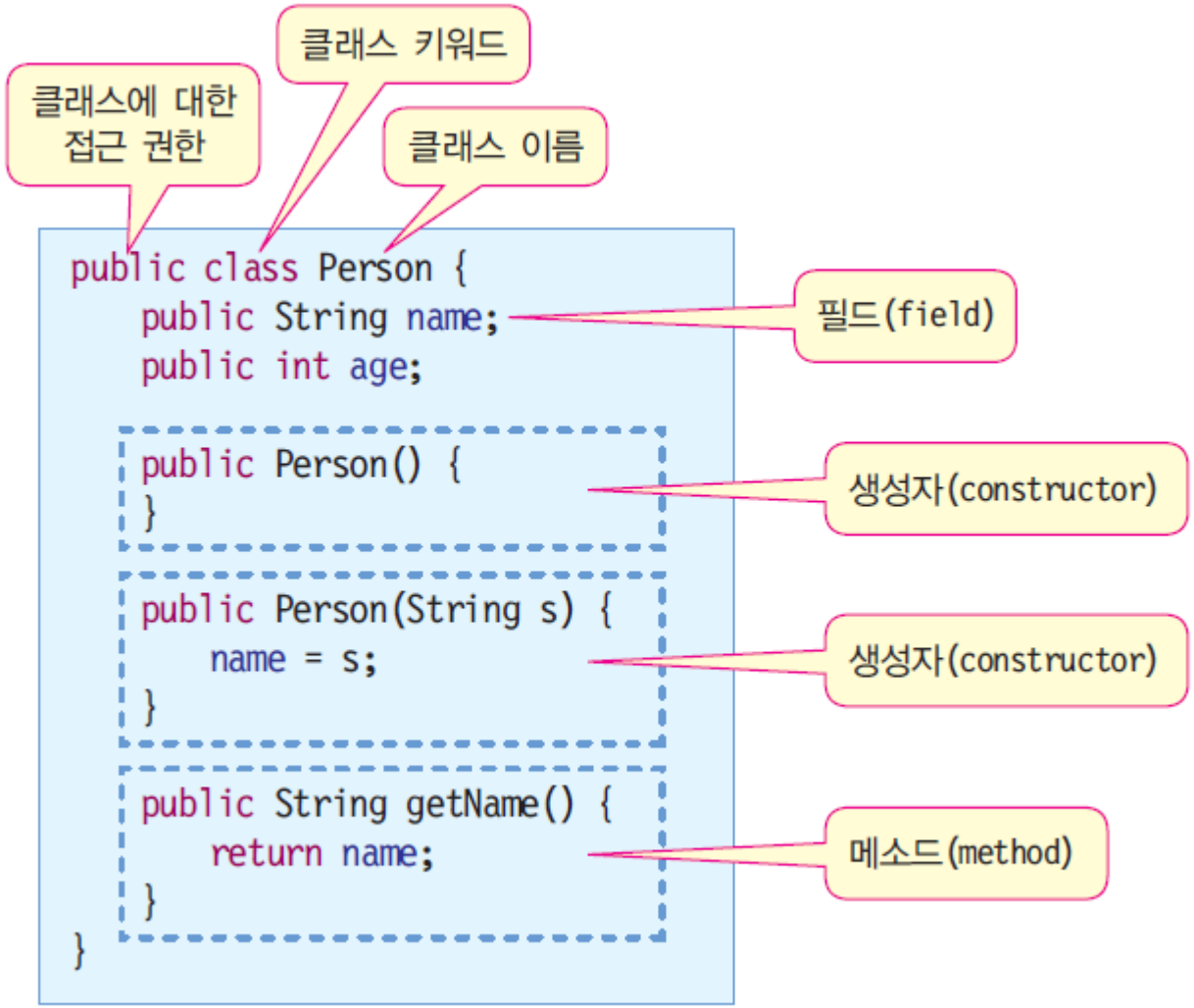
이름 이미녀
직업 골프선수
나이 28
성별 여
혈액형 O

객체 : 이미녀

이름 김미남
직업 교수
나이 47
성별 남
혈액형 AB

객체:김미남

클래스 구성


클래스 키워드
클래스에 대한 접근 권한
클래스 이름
public class Person {
public String name;
public int age;
public Person() {
}
public Person(String s) {
name = s;
}
public String getName() {
return name;
}
}
필드(field)
생성자(constructor)
생성자(constructor)
메소드(method)

클래스 선언

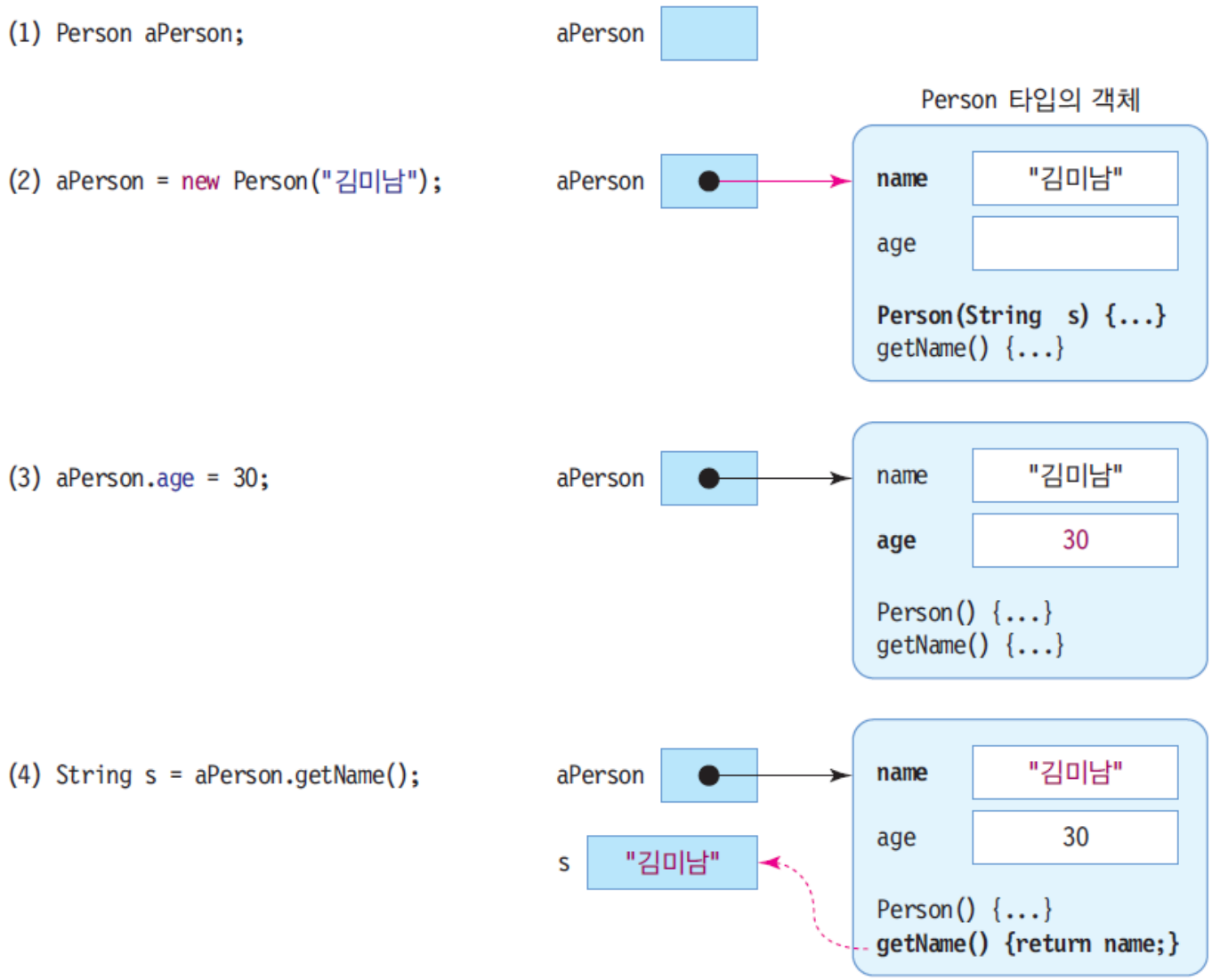
- 클래스 접근 권한, public
  - public 접근 권한은 다른 클래스들이 이 클래스에 대해 사용 혹은 접근이 가능함을 의미
- class Person
  - Person이라는 이름의 클래스 정의
  - class 다음에 클래스의 이름을 선언
  - 클래스는 {로 시작하여 }로 닫으며 이곳에 모든 멤버 필드와 메소드 구현
- 필드(field)
  - 값을 저장할 멤버 변수를 선언
  - 멤버 변수 혹은 필드라고 함
  - 필드 앞에 붙은 접근 지정자 public
    - 이 필드가 다른 클래스에서 접근될 수 있도록 공개한다는 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 public
    - 메소드가 다른 클래스에서 접근될 수 있도록 공개한다는 의미

객체 생성

- 객체 생성
  - 객체는 new 키워드를 이용하여 생성
    - new는 객체의 생성자 호출
- 객체 생성 과정
  1. 객체에 대한 레퍼런스 변수 선언
  2. 객체 생성

```
public static void main (String args[]) {
        Person aPerson;                    // 레퍼런스 변수 aPerson 선언
        aPerson = new Person("김미남");     // Person 객체 생성

        aPerson.age = 30;                  // 객체 멤버 접근
        int i = aPerson.age;               // 30
        String s = aPerson.getName();      // 객체 메소드 호출
}
```

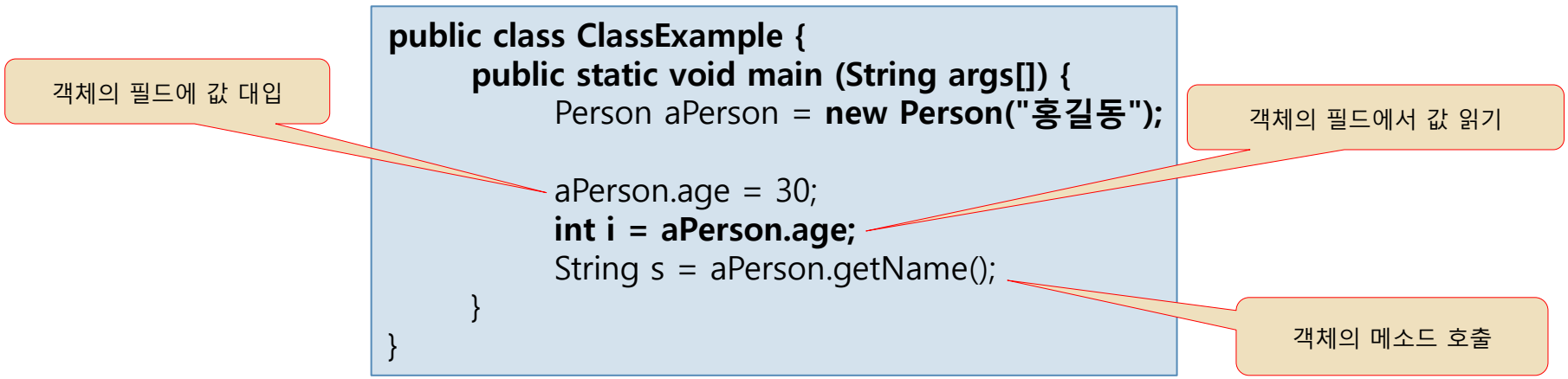
객체 생성 및 사용 예


(1) Person aPerson;
aPerson
Person 타입의 객체
(2) aPerson = new Person("김미남");
aPerson
name
"김미남"
age
Person(String s) {...}
getName() {...}
(3) aPerson.age = 30;
aPerson
name
"김미남"
age
30
Person() {...}
getName() {...}
(4) String s = aPerson.getName();
aPerson
s
"김미남"
name
"김미남"
age
30
Person() {...}
getName() {return name;}

객체의 활용

- 객체의 멤버 접근

객체레퍼런스 . 멤버



예제 4-1 : 상품 하나를 표현하는 클래스 Goods 만들기

상품 하나를 표현하는 클래스 Goods를 작성하라. 상품은 String 타입의 name, int 타입의 price, numberOfStock, sold 등 네 개의 필드를 갖는다. Goods 클래스 내에 main() 메소드를 작성하여 Goods 객체를 하나 생성하고 이 객체에 대한 레퍼런스 변수 명을 camera로 하라. 그리고 나서 camera의 상품 이름(name 필드)을 "Nikon", 값(price)을 400000, 재고 갯수(numberOfStock)를 30, 팔린 개수(sold)를 50으로 설정하라. 그리고 설정된 이들 값을 화면에 출력하라.

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

    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);
    }
}
```

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

예제 4-2 : 지수 클래스 MyExp 만들기

클래스 MyExp를 작성하라. MyExp는 지수값을 표현하는 클래스로서 두 개의 정수형 멤버 필드 base와 exp를 가진다. 2^3의 경우 base는 2이며, exp는 3이 된다. base와 exp는 양의 정수만을 가지는 것으로 가정한다. 또한 MyExp는 정수값을 리턴하는 getValue()라는 멤버 메소드를 제공한다. getValue()는 base와 exp 값으로부터 지수를 계산하여 정수 값으로 리턴한다. 예를 들어 MyExp객체의 base 필드가 2이고 exp가 3이라면 getValue()는 8을 리턴한다.

```
public class MyExp {
    int base;
    int exp;
    int getValue() {
        int res=1;
        for(int i=0; i<exp; i++)
            res = res * base;
        return res;
    }
    public static void main(String[] args) {
        MyExp number1 = new MyExp();
        number1.base = 2;
        number1.exp = 3;
        MyExp number2 = new MyExp();
        number2.base = 3;
        number2.exp = 4;

        System.out.println("2의 3승 = " + number1.getValue());
        System.out.println("3의 4승 = " + number2.getValue());
    }
}
```

```
2의 3승 = 8
3의 4승 = 81
```

객체 배열

- 객체 배열 생성 및 사용

```
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

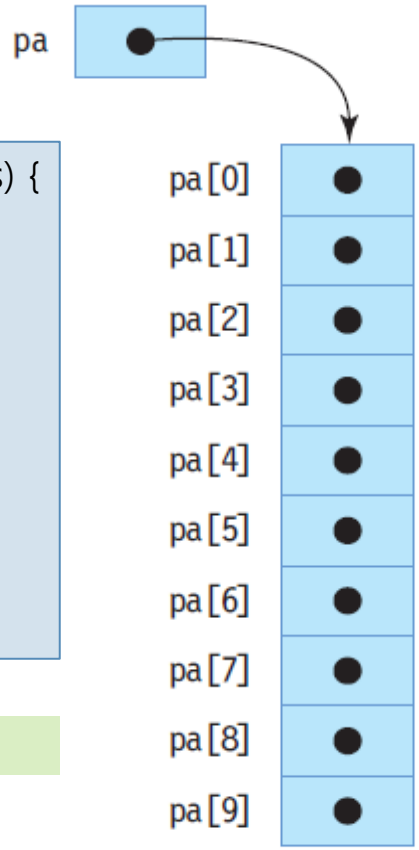
```
for(int i=0; i<pa.length; i++) // 배열 pa의 모든 원소 객체의 age를 출력한다.
   System.out.print(pa[i].age + " ");
```

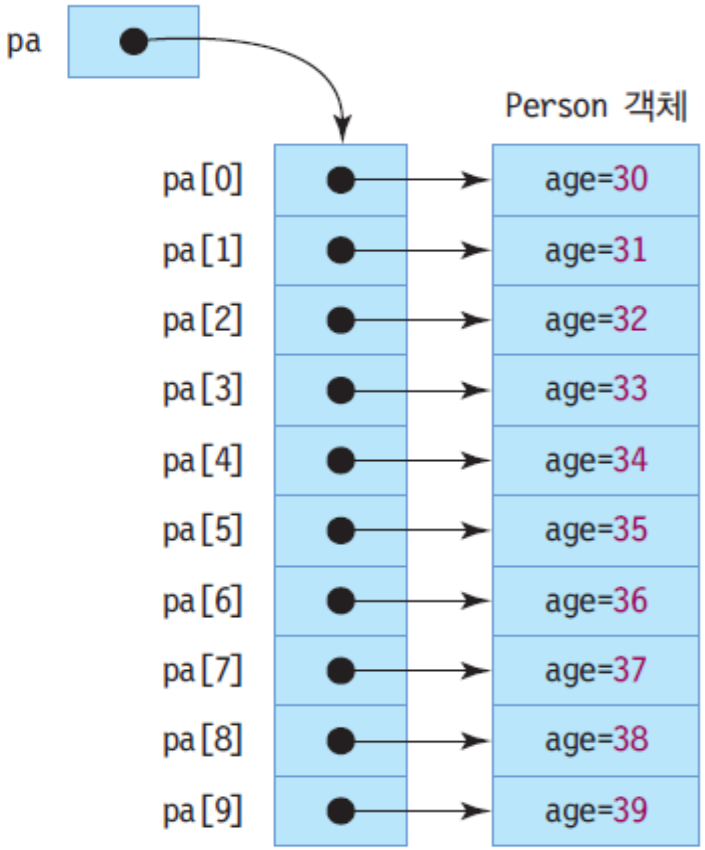
객체 배열 선언과 생성 사례


```
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
```

예제 4-3 : 객체 배열 생성

java.util.Scanner 클래스를 이용하여 상품을 입력 받아 Goods 객체를 생성하고 이들을 Goods 객체 배열에 저장하라. 상품 즉 Goods 객체를 3개 입력 받으면 이들을 모두 화면에 출력하라.

```
import java.util.Scanner;

public class GoodsArray {
    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());
        }
    }
}
```

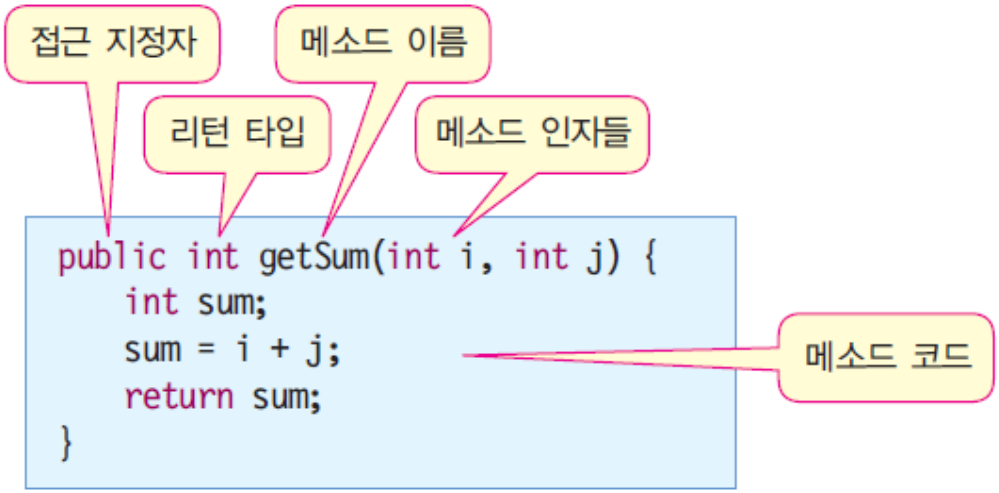
```
class Goods {
    private String name;
    private int price;
    private int numberOfStock;
    private int sold;

    Goods(String n, int p, int nStack, int s) {
        name = n;
        price = p;
        numberOfStock = nStack;
        sold = s;
    }

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

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

키 입력 부분

메소드 형식

- 메소드
  - 메소드는 함수이며 함수 만드는 방법과 동일하게 작성
  - 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

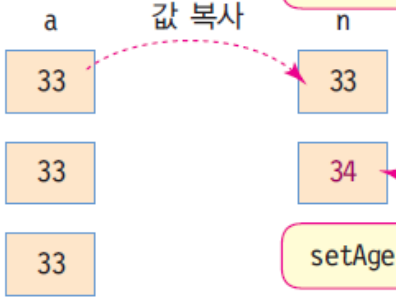


인자 전달 - call by value

- 자바의 인자 전달 방식
  - 값에 의한 호출(call by value)
  - 기본 타입의 값을 전달하는 경우
    - 값이 복사되어 전달
    - 매개 변수 값이 변경되어도 호출한 인자 값은 변경되지 않음
  - 객체 혹은 배열을 전달하는 경우
    - 객체나 배열의 레퍼런스 만 전달됨
      - 객체 혹은 배열이 통째로 복사되어 전달되는 것이 아님
    - 매개 변수와 호출한 실인자가 서로 객체 혹은 배열 공유

call by value : 기본 데이터의 값 전달 사례


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

        aPerson.setAge(a);

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




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

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

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

setAge()가 호출되면 매개변수 n이 생성된다.

setAge()가 끝나면 n은 사라진다.

33

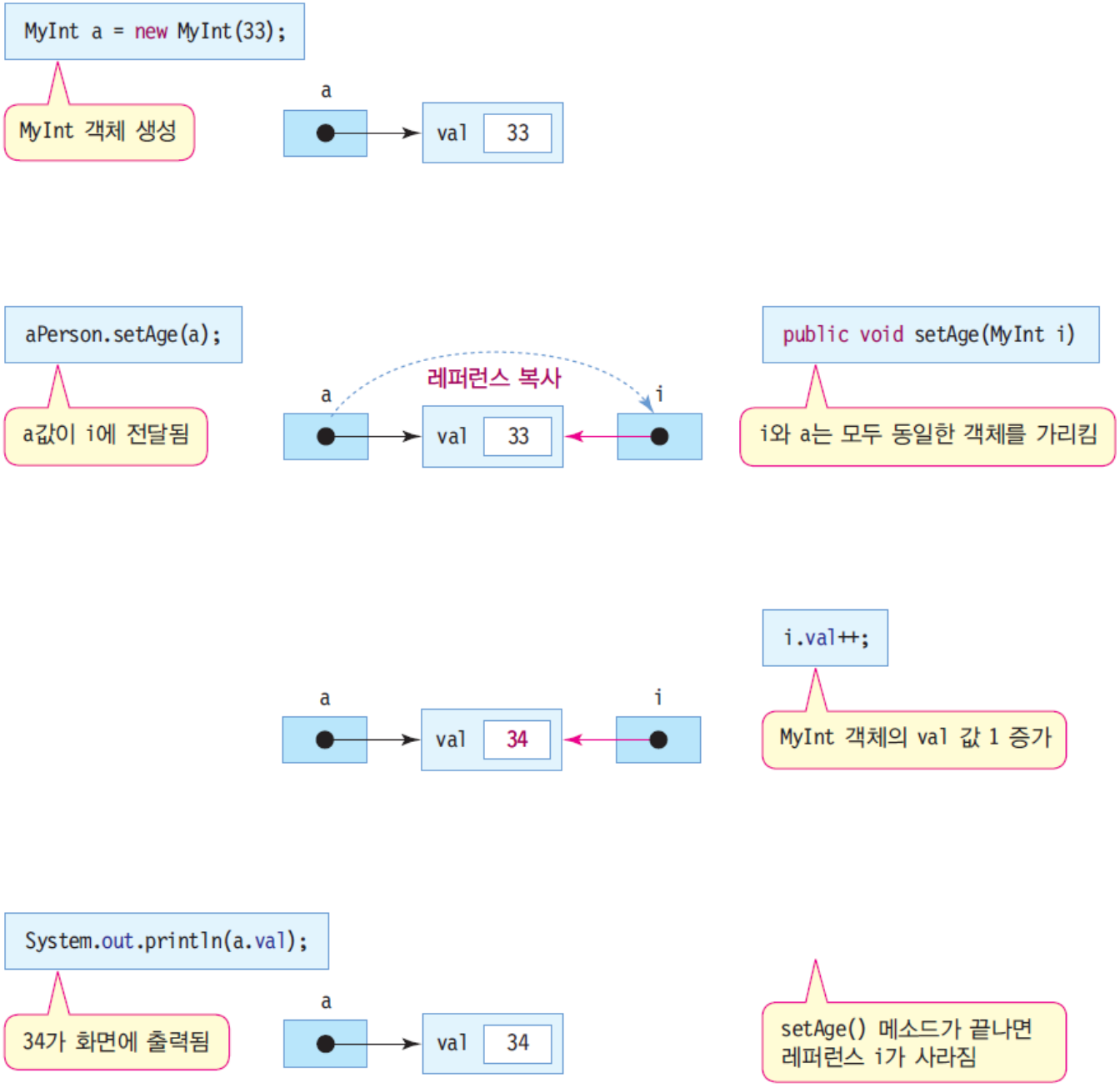
call by value : 객체 전달 사례


```
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);
    }
}
```

호출

```
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++;
    }

}
```

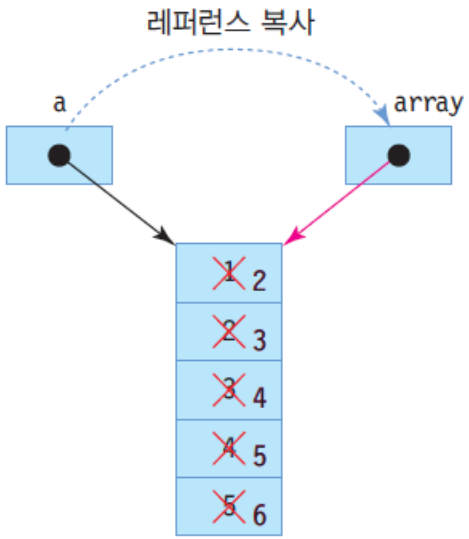

* 객체가 복사되어 전달되는 것이 아님
객체에 대한 레퍼런스 만이 복사되어 전달


MyInt a = new MyInt(33);
MyInt 객체 생성
a
val
33
aPerson.setAge(a);
a값이 i에 전달됨
레퍼런스 복사
a
i
val
33
public void setAge(MyInt i)
i와 a는 모두 동일한 객체를 가리킴
i.val++;
MyInt 객체의 val 값 1 증가
a
i
val
34
System.out.println(a.val);
34가 화면에 출력됨
a
val
34
setAge() 메소드가 끝나면
레퍼런스 i가 사라짐

call by value : 배열 전달 사례

- 인자로 배열을 전달하면 배열의 레퍼런스만이 전달됨

```
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]+" ");
    }
```




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

2 3 4 5 6

예제 4-4 : 배열의 전달

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

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

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

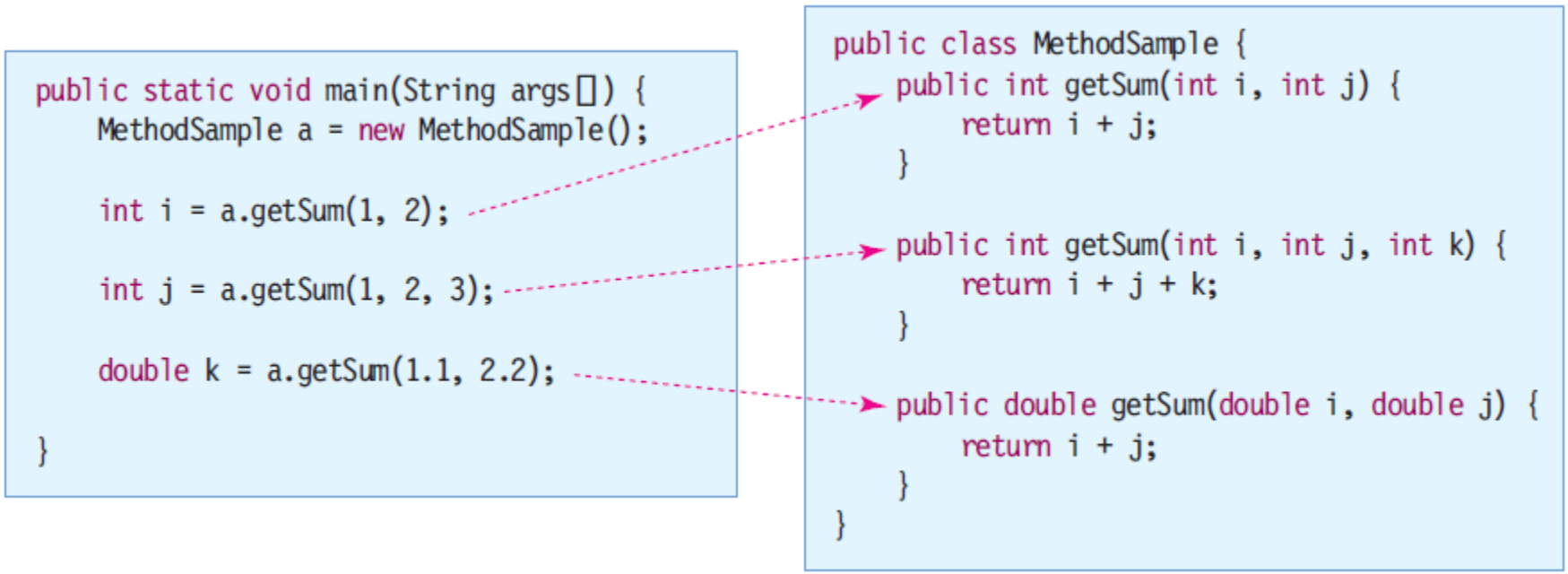
메소드 오버로딩

- 오버로딩(Overloading)
  - 한 클래스 내에서 두 개 이상의 이름이 같은 메소드 작성
    - 메소드 이름이 동일하여야 함
    - 인자가 개수 서로 다르거나, 인자 타입이 서로 달라야 함
    - 이름이 같고 인자의 개수나 타입이 모두 같은데, 리턴 타입이 다르면 오버로딩이 성립되지 않으며, 컴파일 오류 발생

```
// 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


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;
}
}

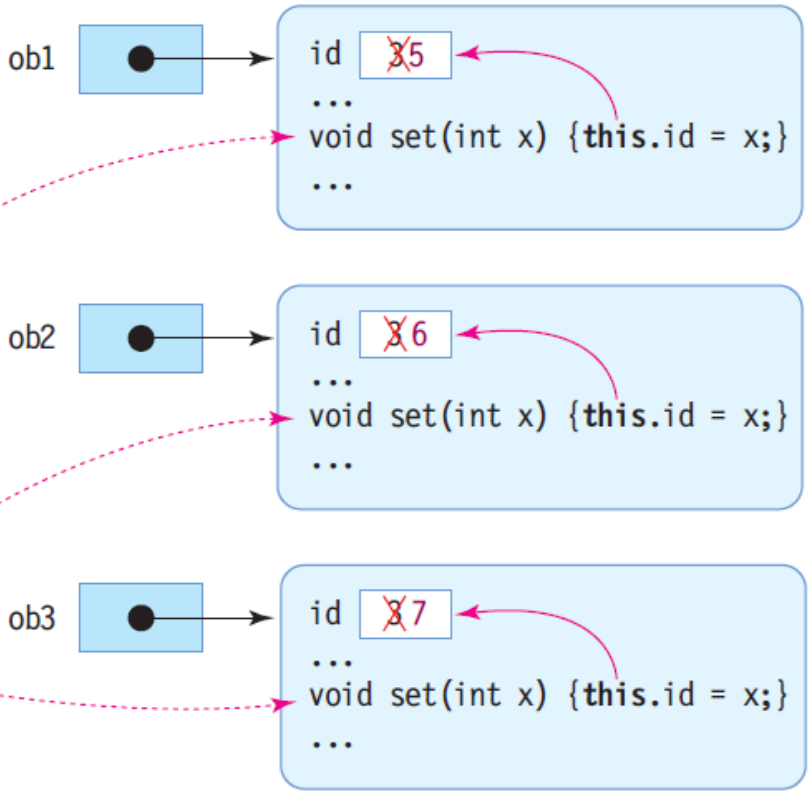
this 레퍼런스

- 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

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

this에 대한 이해

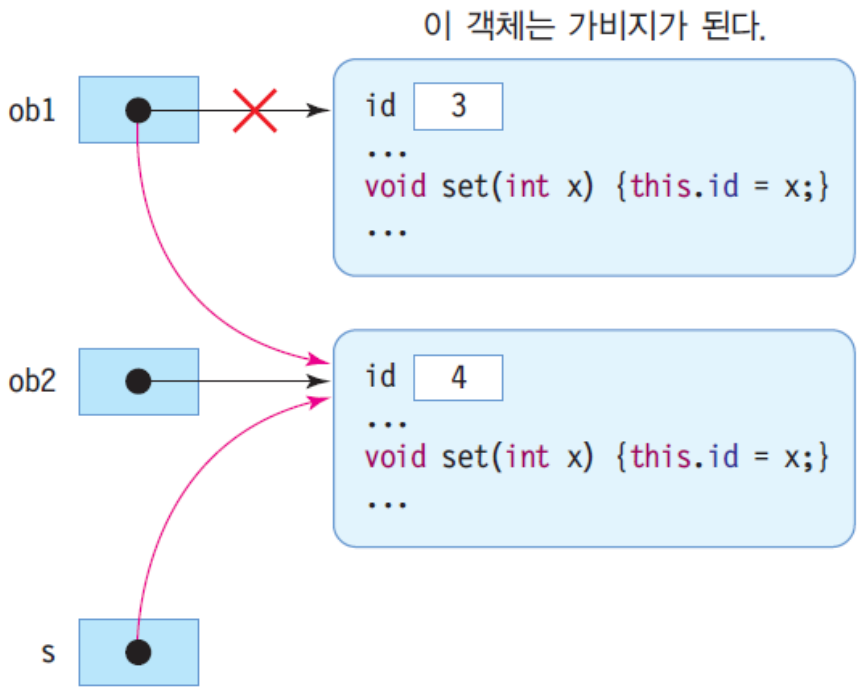

```
public class Samp {
    int id;
    public Samp(int x) {this.id = x;}
    public void set(int x) {this.id = x;}
    public int get() {return this.id;}

    public 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);
    }
}
```



객체의 치환

* 객체의 치환은 객체가 복사되는 것이 아니며 레퍼런스가 복사된다.

```
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 static void main(String [] args) {
        Samp ob1 = new Samp(3);
        Samp ob2 = new Samp(4);
        Samp s;

        s = ob2;
        ob1 = ob2; // 객체의 치환
        System.out.println("ob1.id="+ob1.id);
        System.out.println("ob2.id="+ob2.id);
    }
}
```

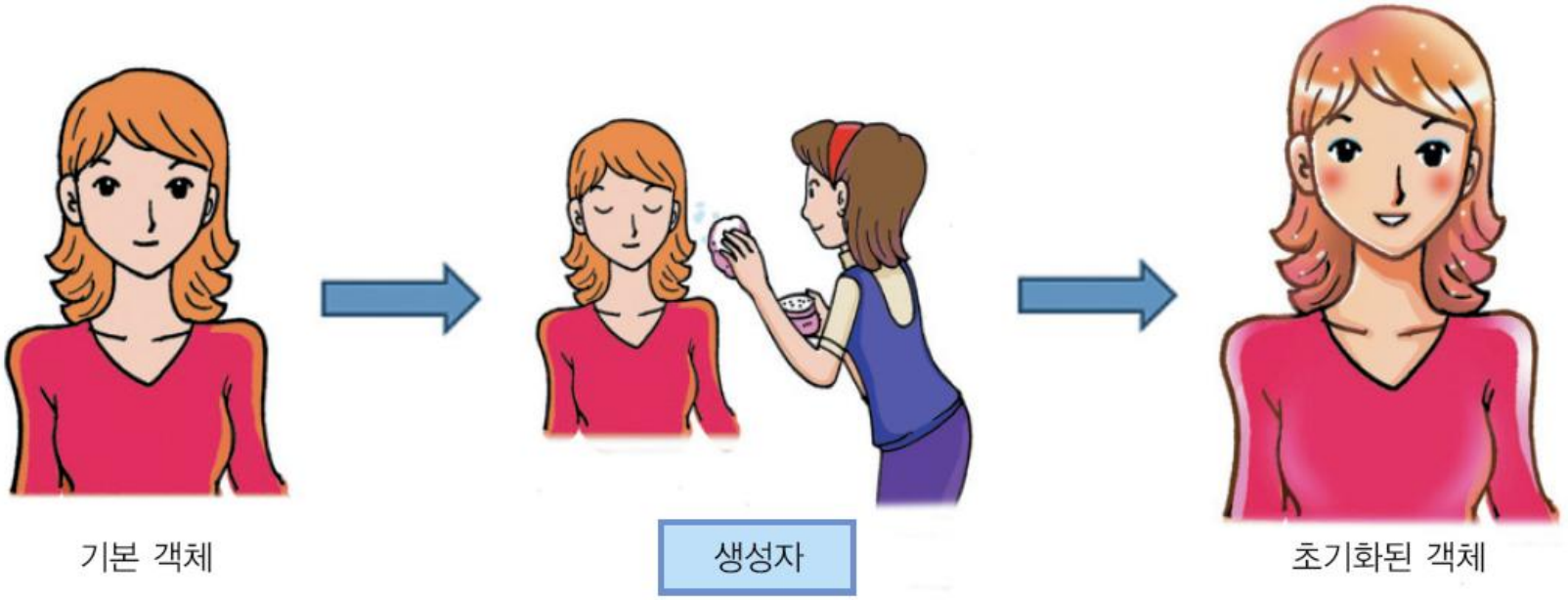


ob1.id=4
ob2.id=4

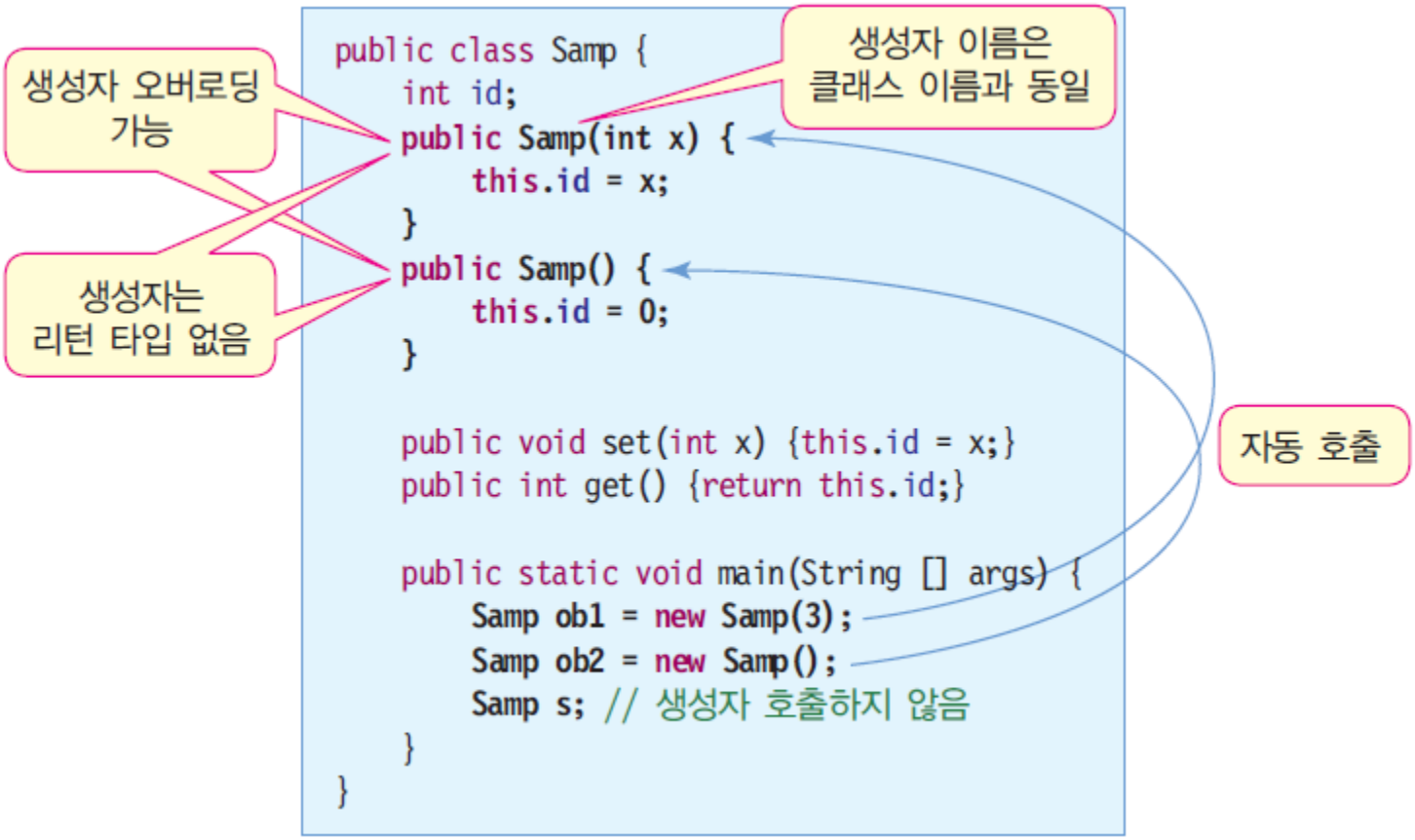
생성자

- 생성자의 특징
  - 생성자는 메소드
  - 생성자 이름은 클래스 이름과 동일
  - 생성자는 new를 통해 객체를 생성할 때만 호출됨
  - 생성자도 오버로딩 가능
  - 생성자는 리턴 타입을 지정할 수 없다.
  - 생성자는 하나 이상 선언되어야 함
    - 개발자가 생성자를 정의하지 않으면 자동으로 기본 생성자가 정의됨
      - 컴파일러에 의해 자동 생성
      - 기본 생성자를 디폴트 생성자(default constructor)라고도 함

생성자 개념



생성자 정의와 생성자 호출



예제 4-5 : 생성자 정의와 호출

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;
   }

   public static void main(String [] args) {
      Book javaBook = new Book("Java JDK", "황기태", 3333);
   }
}
```

기본 생성자

- 기본 생성자(default constructor)
  - 클래스에 생성자가 하나도 선언되지 않은 경우
  - 컴파일러에 의해 자동으로 생성
    - 인자 없는 생성자
    - 아무 작업 없이 단순 리턴
  - 디폴트 생성자라고도 부름

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

    public static void main(String [] args) {
        DefaultConstructor p = new DefaultConstructor();
        p.setX(3);
    }
}
```

개발자가 작성한 코드

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

    public DefaultConstructor() { }

    public static void main(String [] args) {
        DefaultConstructor p= new DefaultConstructor();
        p.setX(3);
    }
}
```

컴파일러에 의해 자동 삽입된 기본 생성자

컴파일러가 자동으로 기본 생성자를 삽입한 코드

기본 생성자가 자동 생성되지 않는 경우

- 클래스에 생성자가 하나라도 존재하면 자동으로 기본 생성자가 생성되지 않음

```
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 static void main(String [] args) {
      DefaultConstructor p1= new DefaultConstructor(3);
      int n = p1.getX();

      DefaultConstructor p2= new DefaultConstructor();
      p2.setX(5);
   }
}
```

컴파일러가 기본 생성자를 자동 생성하지 않음

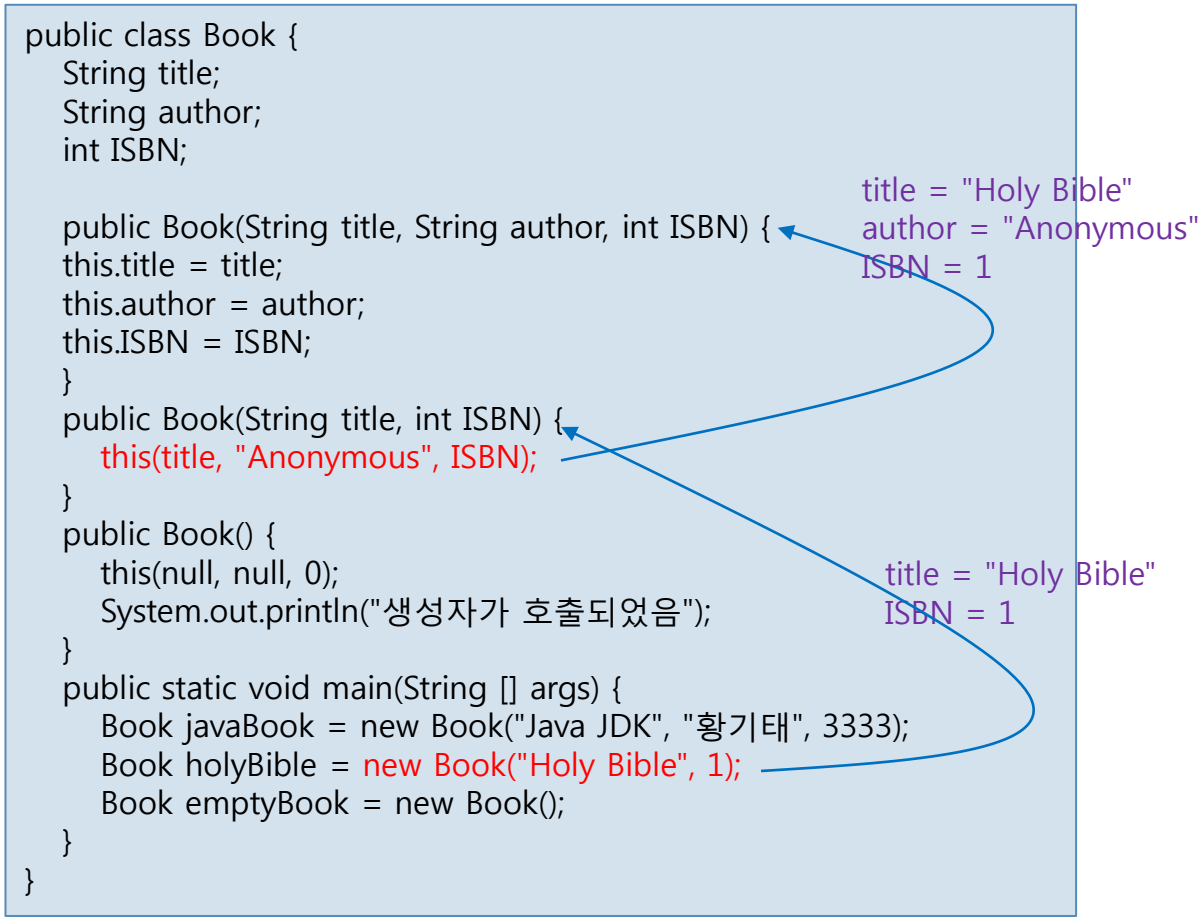
public DefaultConstructor() { }

컴파일 오류.
해당하는 생성자가 없음 !!!

this(), 생성자에서 다른 생성자 호출

- this()
  - 같은 클래스의 다른 생성자 호출
  - 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 static void main(String [] args) {
        Book javaBook = new Book("Java JDK", "황기태", 3333);
        Book holyBible = new Book("Holy Bible", 1);
        Book emptyBook = new Book();
    }
}
```



this() 사용 실패 예

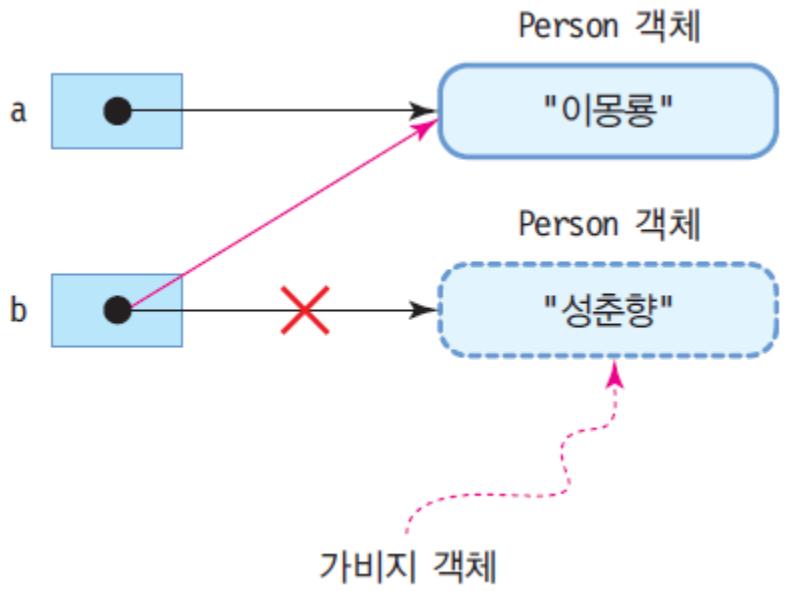

```
public Book() {
   System.out.println("생성자가 호출되었음");
   this(null, null, 0); // 생성자의 첫 번째 문장이 아니기 때문에 컴파일 오류
}
```

객체의 소멸과 가비지

- 객체 소멸
  - new에 의해 생성된 객체 메모리를 자바 가상 기계에게 되돌려 주는 행위
  - 소멸된 객체 공간은 가용 메모리에 포함
- 자바는 객체 삭제 기능 없음
  - 개발자에게는 매우 다행스러운 기능
    - C/C++에서는 할당 받은 객체를 개발자가 프로그램 내에서 삭제해야 함
- 가비지
  - 가비지
    - 자신에 대한 레퍼런스가 없는 객체
  - 가비지 켈렉션
    - 자바 가상 기계의 가비지 컬렉터가 자동으로 가비지를 수집하여 반환

가비지 사례

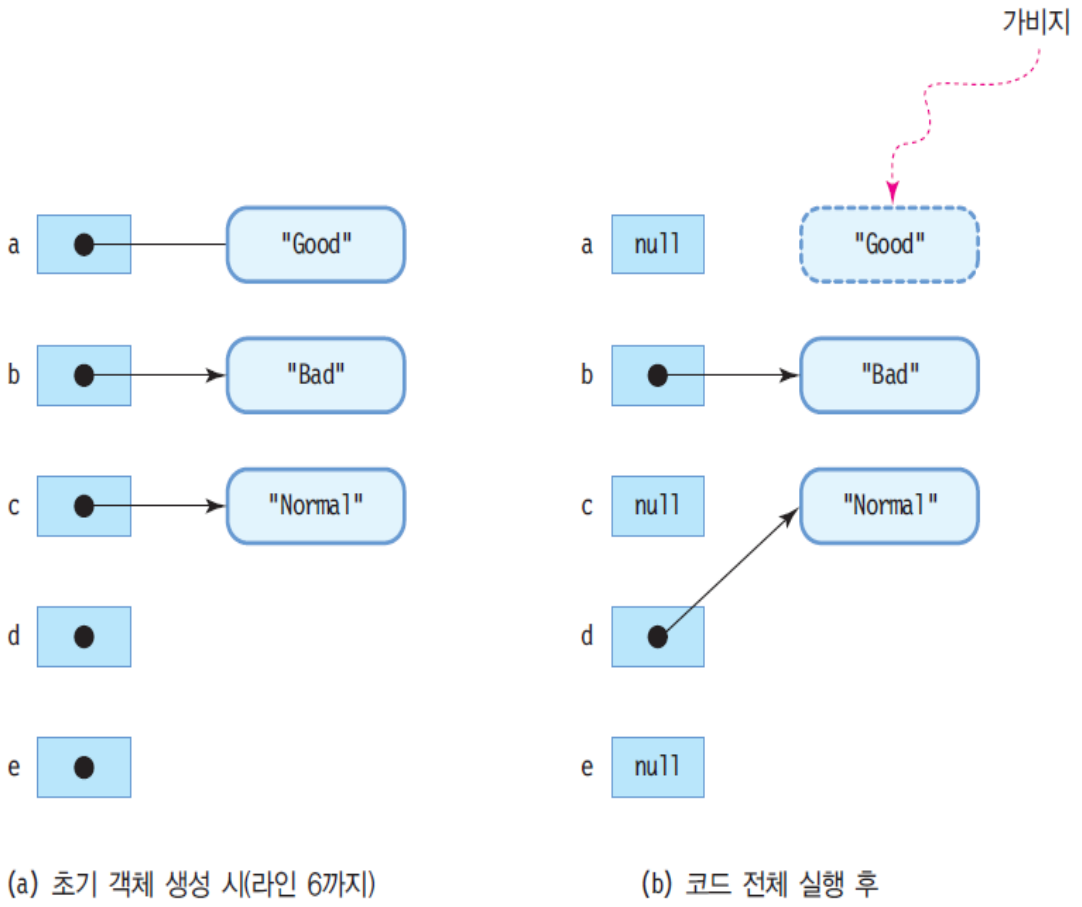

```
Person a, b;
a = new Person("이몽룡");
b = new Person("성춘향");
b = a; // b가 가리키던 객체는 가비지가 됨
```



예제 4-6 : 가비지 발생

다음 소스에서 언제 가비지가 발생하는지 설명하라.

```
public class GarbageEx {
    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 생성 시(라인 6까지)

(b) 코드 전체 실행 후

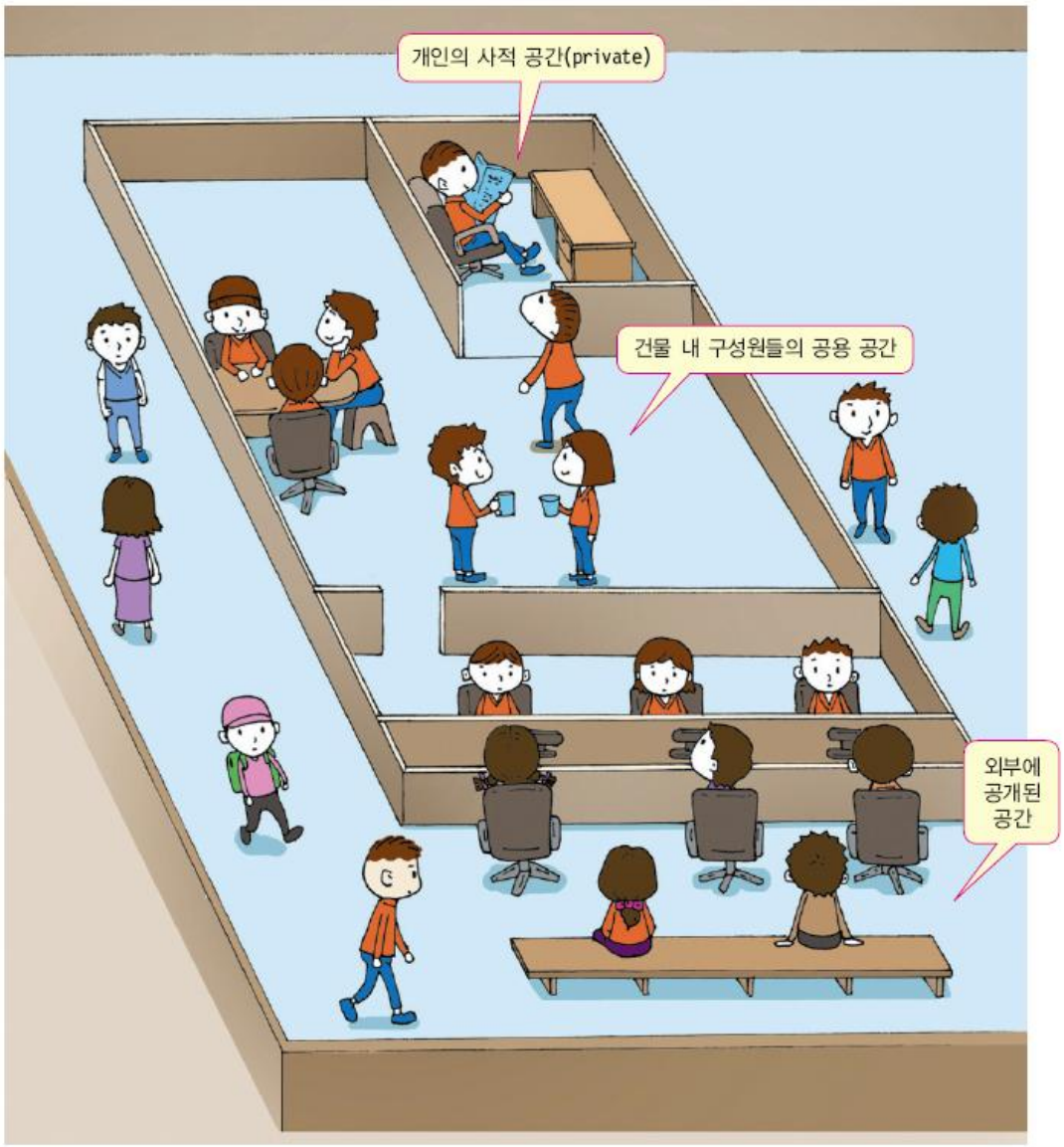
가비지 컬렉션

- 가비지 컬렉션
  - 자바에서는 가비지 자동 회수
    - 가용 메모리 공간으로 확보
  - 가비지 컬렉터(garbage collector)에 의해 자동 수행
- 개발자에 의한 강제 가비지 컬렉션
  - System 또는 Runtime 객체의 gc() 메소드 호출

```
System.gc(); // 가비지 컬렉션 작동 요청
```

    - 자바 가상 기계에 강력한 가비지 컬렉션을 요청
      - 그러나 자바 가상 기계가 가비지 컬렉션 시점을 전적으로 판단

접근 지정자 이해



클래스 접근 지정자

- 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
      - 다른 말로 같은 디렉토리에 있는 클래스끼리 접근 가능

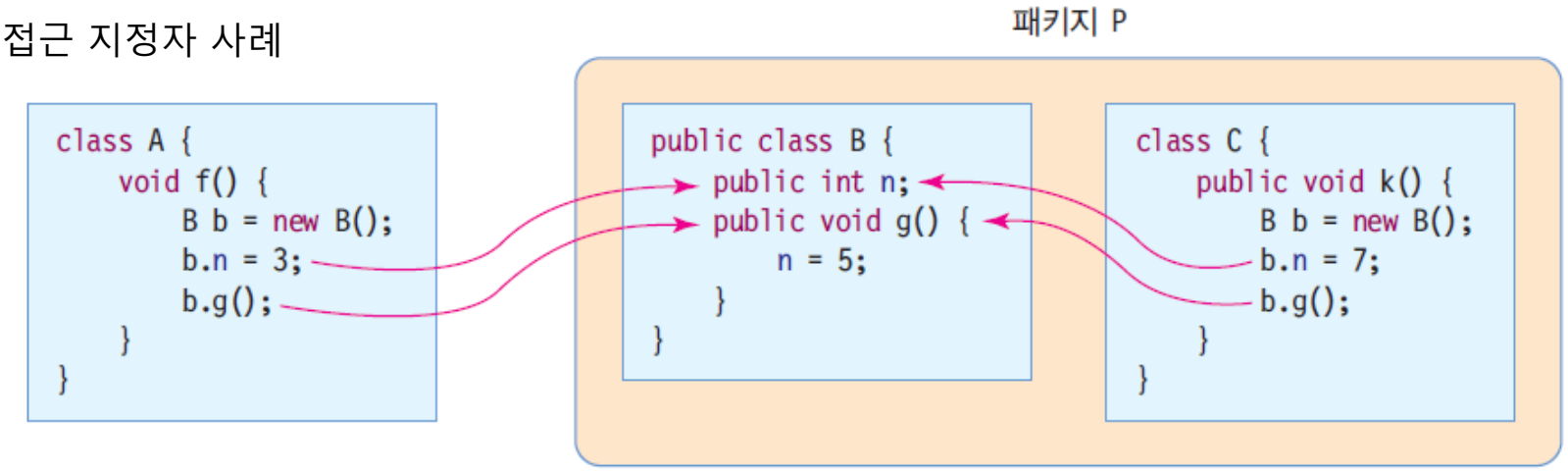
멤버 접근 지정자

default (또는 package-private)	**•같은 패키지 내에서 접근 가능**
public	•패키지 내부, 외부 클래스에서 접근 가능
private	•정의된 클래스 내에서만 접근 가능 •상속 받은 하위 클래스에서도 접근 불가
protected	•같은 패키지 내에서 접근 가능 •다른 패키지에서 접근은 불가하나 상속을 받은 경우 하위 클래스에서는 접근 가능

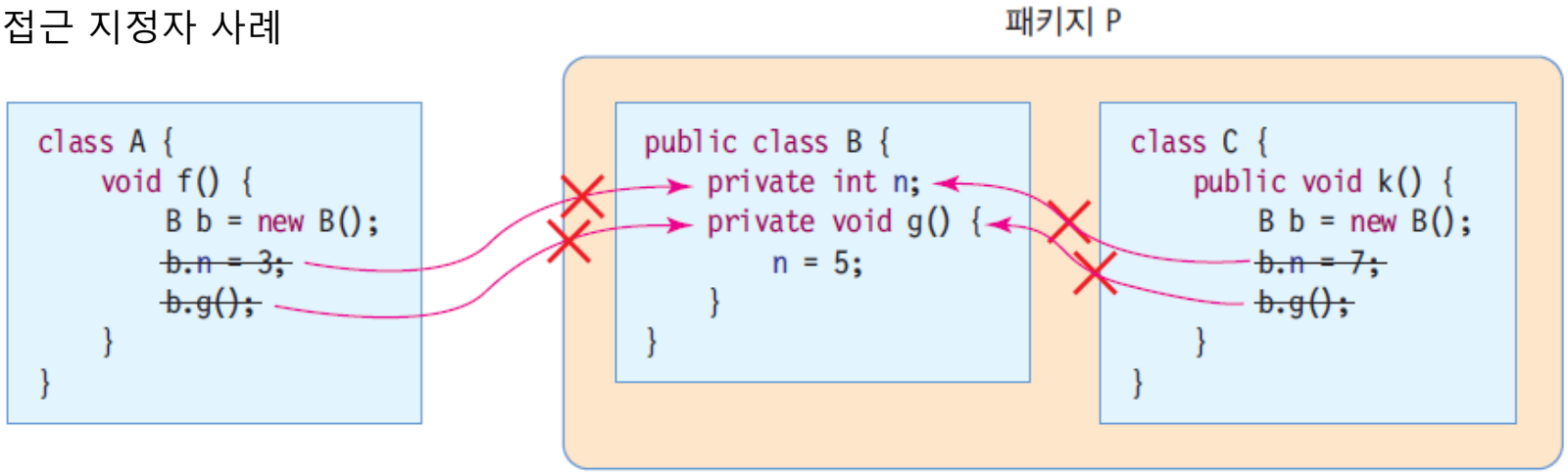
멤버에 접근하는 클래스	**멤버의 접근 지정자**			
	default	private	protected	public
같은 패키지의 클래스	O	X	O	O
다른 패키지의 클래스	X	X	X	O

접근 지정자의 이해

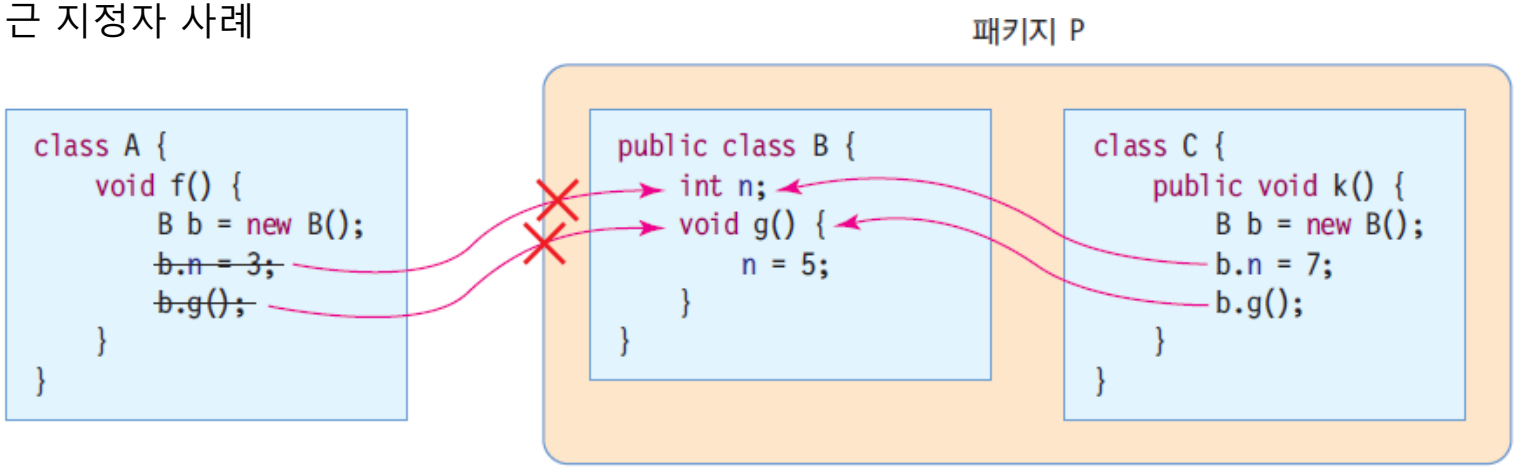
public 접근 지정자 사례



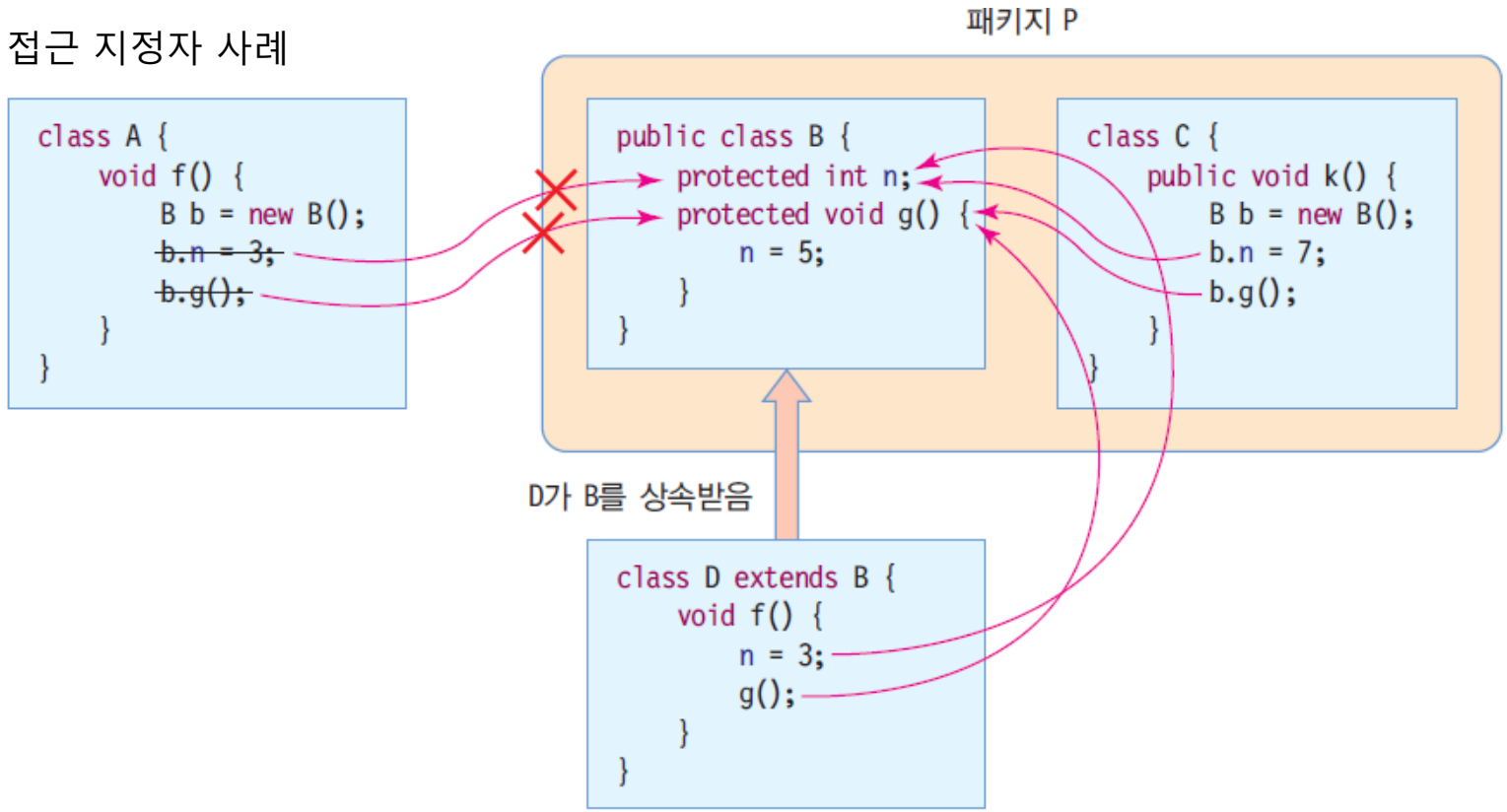
private 접근 지정자 사례



default 접근 지정자 사례



protected 접근 지정자 사례



예제 4-7 : 접근 지정자의 사용

다음의 소스를 컴파일 해보고 오류가 난 이유를 설명하고 오류를 수정하시오.

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

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

- Sample 클래스의 a와 c는 각각 public, default 지정자로 선언이 되었으므로, 같은 패키지에 속한 AccessEx 클래스에서 접근 가능
- b는 private으로 선언이 되었으므로 AccessEx 클래스에서 접근 불가능

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

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

예제 4-7 결과

오류가 수정된 소스

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

- private 접근 지정자를 갖는 멤버 b를 위해 클래스 내부에 getB()/setB() 메소드 만들어 접근

static 이해를 위한 그림

눈은 각 사람마다 있고 공기는 모든 사람이 소유(공유)한다

사람은 모두 각각 눈을 가지고 태어난다.

세상에는 이미 공기가 있으며 태어난 사람은 모두 공기를 공유한다.
그리고 공기 역시 각 사람의 것이다.

static 멤버와 non-static 멤버

- non-static 멤버의 특성
  - 공간적 - 멤버들은 객체마다 독립적으로 별도 존재
    - 인스턴스 멤버라고도 부름
  - 시간적 - 필드와 메소드는 객체 생성 후 비로소 사용 가능
  - 비공유의 특성 - 멤버들은 여러 객체에 의해 공유되지 않고 배타적
- static 멤버란?
  - 객체를 생성하지 않고 사용가능
  - 객체마다 생기는 것이 아님
  - 클래스당 하나만 생성됨
    - 클래스 멤버라고도 부름
  - 특성
    - 공간적 특성 - static 멤버들은 클래스 당 하나만 생성.
    - 시간적 특성 - static 멤버들은 클래스가 로딩될 때 공간 할당.
    - 공유의 특성 - static 멤버들은 동일한 클래스의 모든 객체에 의해 공유

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

   static int m; // static 필드
   static void f() {...} // static 메소드
}
```

non-static 멤버와 static 멤버의 차이

	non-static 멤버	**static 멤버**

선언	class Sample {
	  int n;
	  void g() {...}
	}
		class Sample {
		  static int m;
		  static void g() {...}
		}

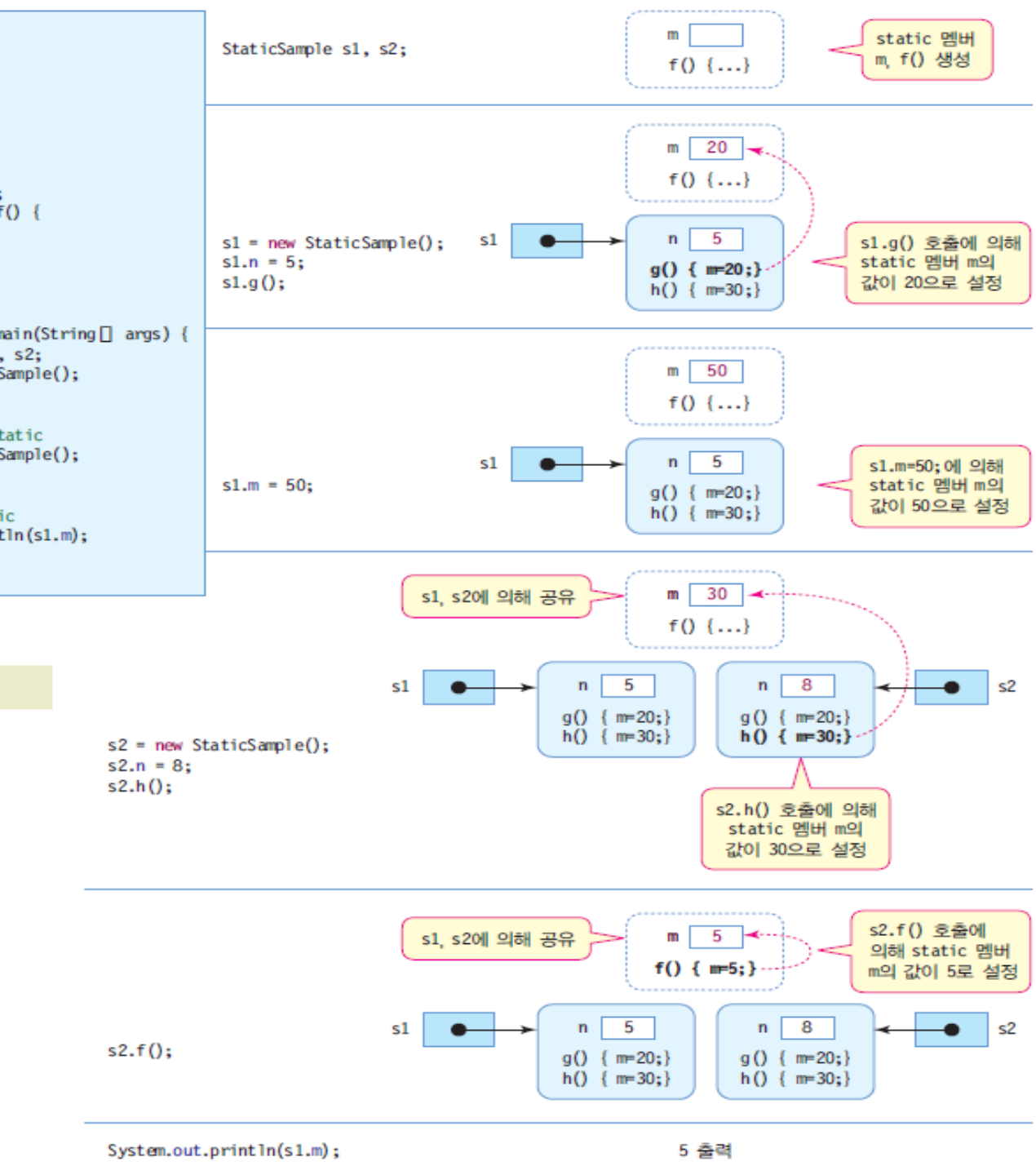
공간적 특성	멤버는 객체마다 별도 존재. - 인스턴스 멤버라고 부름.	멤버는 클래스 당 하나 생성 - 멤버는 객체 내부가 아닌 별도의 공간에 생성 - 클래스 멤버라고 부름
시간적 특성	객체 생성 시 함께 멤버 생성됨 - 객체가 생길 때 멤버도 생성 - 객체 생성 후 멤버 사용 가능 - 객체가 사라지면 멤버도 사라짐	클래스 로딩 시에 멤버 생성 - 객체가 생기기 전에 이미 생성 - 객체가 생기기 전에도 사용 가능 - 객체가 사라져도 멤버는 사라지지 않음 - 멤버는 프로그램이 종료될 때 사라짐
공유의 특성	동일한 클래스의 객체들에 의해 공유되지 않음. - 멤버는 객체 내에 각각 공간 유지	동일한 클래스의 객체들에 의해 공유됨

static 멤버를 객체의 멤버로 접근하는 사례

```
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 Ex {
    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);
    }
}
```

실행 결과

```
5
```



static 멤버를 클래스 이름으로 접근하는 사례

```
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 Ex {
    public static void main(String[] args) {
        StaticSample.m = 10;

        StaticSample s1;
        s1 = new StaticSample();
        System.out.println(s1.m);
        s1.f();
        StaticSample.f();
    }
}
```

실행 결과

```
10
```

static의 활용

- 전역 변수와 전역 함수를 만들 때 활용
  - 자바의 캡슐화 원칙 지킴
    - 다른 클래스에서 공유하는 전역 변수나 전역 함수도 반드시 클래스 내부에 구현해야 함
- static 멤버를 가진 클래스 사례
  - 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

- 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

예제 4-8 : static을 이용한 달러와 우리나라 원화 사이의 변환 예제

static 필드와 메소드를 이용하여 달러와 한국 원화 사이의 변환을 해주는 환율 계산기를 만들어 보자.

```
class CurrencyConverter {
    private static double rate; //  한국 원화에 대한 환율
    public static double toDollar(double won) {
        return won/rate; // 한국 원화를 달러로 변환
    }
    public static double toKWR(double dollar) {
        return dollar * rate; // 달러를 한국 원화로 변환
    }
    public static void setRate(double r) {
        rate = r; // 환율 설정. KWR/$1
    }
}
public class StaticMember {
    public static void main(String[] args) {
        CurrencyConverter.setRate(1121); // 미국 달러 환율 설정
        System.out.println("백만원은 " + CurrencyConverter.toDollar(1000000) + "달러입니다.");
        System.out.println("백달러는 " + CurrencyConverter.toKWR(100) +"원입니다.");
    }
}
```

```
백만원은 892.0606601248885달러입니다.
백달러는 112100.0원입니다.
```

final 클래스와 메소드

- 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

- final 필드, 상수 정의
  - 상수를 정의할 때 사용

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

  - 상수 필드는 선언 시에 초기 값을 지정하여야 한다
  - 상수 필드는 한 번 정의되면 값을 변경할 수 없다

```
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.
        }
}
```