

Revisit Java Programming (Class)

Fall, 2020

Table of Contents

- Class

Object?

- Everything in the world



TV



의자



책



집



카메라



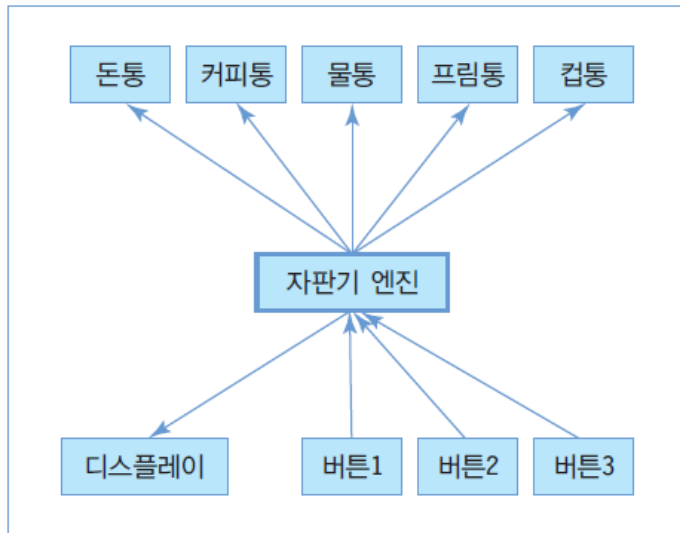
컴퓨터

- Object has status and behavior
- e.g., Camera
 - status: stored pictures
 - behavior: takePicture()

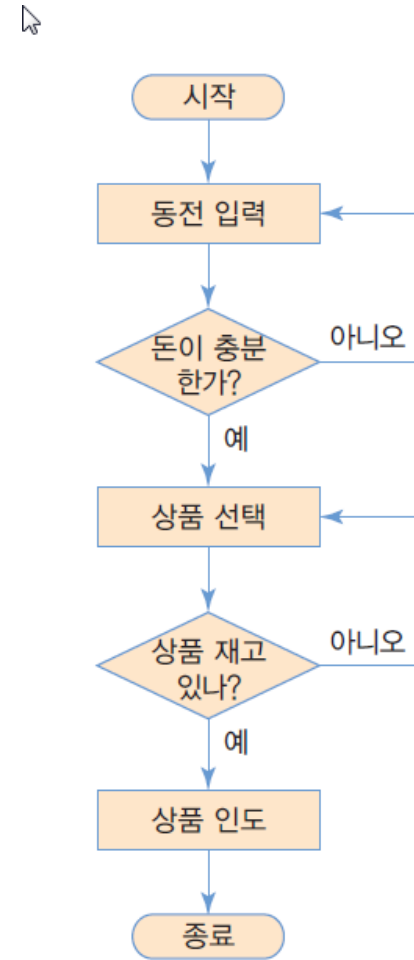
절차 지향 프로그래밍과 객체 지향 프로그래밍

- 절차 지향 프로그래밍
 - 작업 순서 표현
 - 작업을 함수로 작성한, 함수들의 집합
- 객체 지향 프로그래밍
 - 객체들간의 상호 작용으로 표현
 - 클래스 혹은 객체들의 집합으로 프로그램 작성

커피 자판기



객체지향적 프로그래밍의 객체들의 상호 관련성



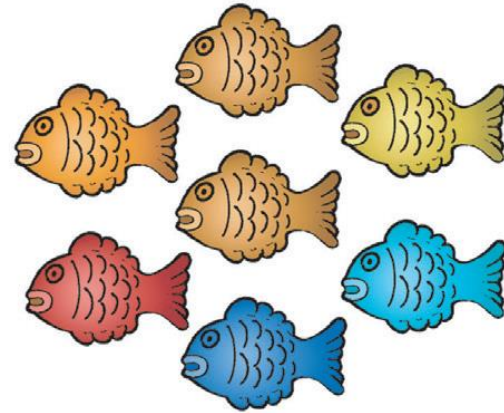
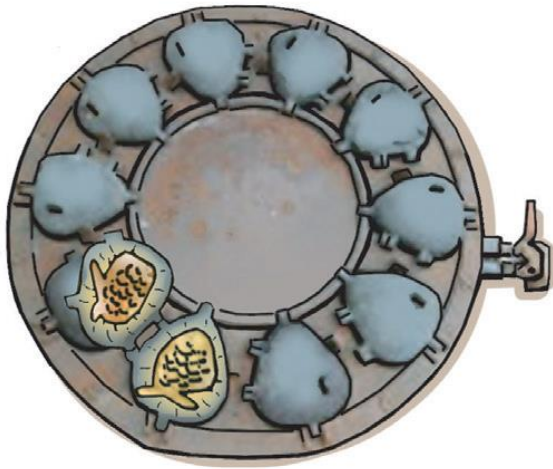
절차지향적 프로그래밍의 실행 절차

클래스와 객체

- 클래스
 - 객체를 만들어내기 위한 설계도 혹은 틀
 - 객체의 속성(state)과 행동(behavior) 포함
- 객체
 - 클래스의 모양 그대로 찍어낸 실체
 - 프로그램 실행 중에 생성되는 실체
 - 메모리 공간을 갖는 구체적인 실체
 - 인스턴스(instance)라고도 부름
- 사례
 - 클래스: 소나타자동차, 객체: 출고된 실제 소나타 100대
 - 클래스: 사람, 객체: 나, 너, 윗집사람, 아랫집사람
 - 클래스: 붕어빵틀, 객체: 구워낸 붕어빵들

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

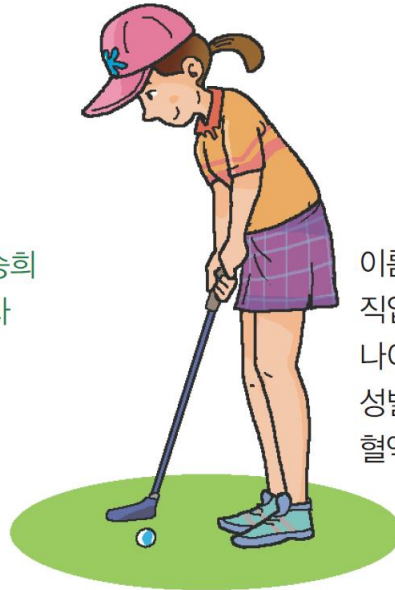
클래스: 사람

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 골프 선수
나이 28
성별 여
혈액형 O

객체: 이미녀



이름 김미남
직업 교수
나이 47
성별 남
혈액형 AB

객체: 김미남

* 객체들은 클래스에 선언된 동일한 속성을 가지지만, 객체마다 서로 다른 고유한 값으로 구분됨

클래스 구성

접근 권한

클래스 선언

클래스 이름

```
public class Circle {
```

```
    public int radius; // 원의 반지름 필드
```

```
    public String name; // 원의 이름 필드
```

필드(변수)

```
    public Circle() { // 원의 생성자 메소드
```

```
    }
```

```
    public double getArea() { // 원의 면적 계산 메소드
```

```
        return 3.14*radius*radius;
```

메소드

```
    }
```

```
}
```


클래스 구성 설명

- 클래스 선언, `class Circle`
 - `class` 키워드로 선언
 - 클래스는 {로 시작하여 }로 닫으며 이곳에 모든 필드와 메소드 구현
 - `class Circle`은 `Circle` 이름의 클래스 선언
 - 클래스 접근 권한, `public`
 - 다른 클래스들에서 `Circle` 클래스를 사용하거나 접근할 수 있음을 선언
- 필드와 메소드
 - 필드 (field) : 객체 내에 값을 저장하는 멤버 변수
 - 메소드 (method) : 함수이며 객체의 행동(행위)를 구현
- 필드의 접근 지정자, `public`
 - 필드나 메소드 앞에 붙어 다른 클래스의 접근 허용을 표시
 - `public` 접근 지정자 : 다른 모든 클래스의 접근 허용
- 생성자
 - 클래스의 이름과 동일한 특별한 메소드
 - 객체가 생성될 때 자동으로 한 번 호출되는 메소드
 - 개발자는 객체를 초기화하는데 필요한 코드 작성

객체 생성 및 접근

- 객체 생성
 - 반드시 **new** 키워드를 이용하여 생성
 - new는 객체의 생성자 호출
- 객체 생성 과정
 - 객체에 대한 레퍼런스 변수 선언
 - 객체 생성
 - 클래스 타입 크기의 메모리 할당
 - 객체 내 생성자 코드 실행
- 객체의 멤버 접근
 - 객체 레퍼런스.멤버

객체 생성과 접근

1. 레퍼런스 변수 선언

(1) Circle pizza;

pizza



2. 객체 생성

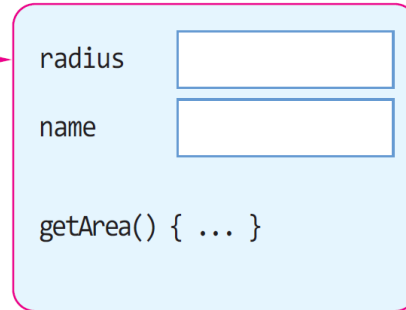
- new 연산자 이용

(2) pizza = new Circle();

pizza



Circle 타입의 객체



객체 메모리
할당 및
객체 생성

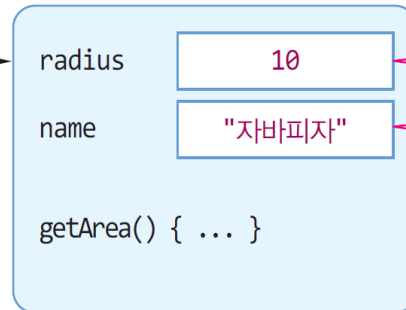
3(4). 객체 멤버 접근

- 점(.) 연산자 이용

(3) pizza.radius = 10;

pizza.name = "자바피자"

pizza



radius 값 변경

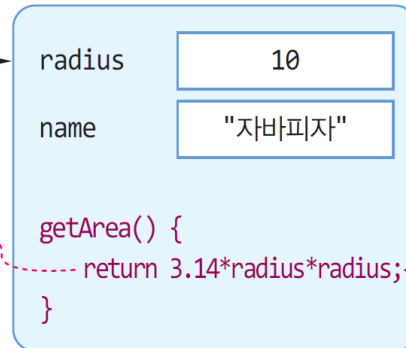
name 값 변경

(4) double area = pizza.getArea();

pizza



area

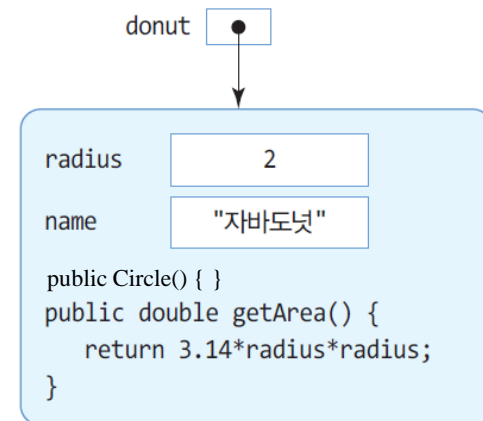
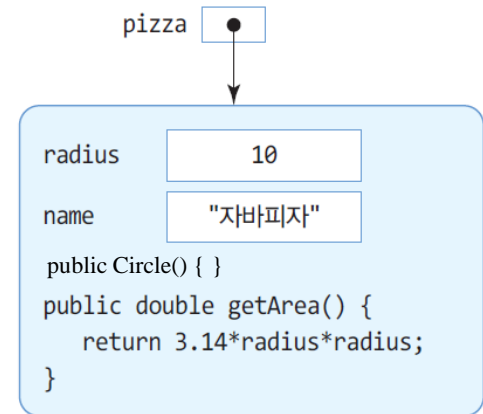


getArea()
메소드 실행

예제 4-1 : Circle 클래스의 객체 생성 및 활용

반지름과 이름을 가진 Circle 클래스를 작성하고, Circle 클래스의 객체를 생성하라. 그리고 객체가 생성된 모습을 그려보라.

```
public class Circle {  
    int radius;           // 원의 반지름 필드  
    String name;          // 원의 이름 필드  
  
    public Circle() { }    // 원의 생성자  
  
    public double getArea() { // 원의 면적 계산 메소드  
        return 3.14*radius*radius;  
    }  
  
    public static void main(String[] args) {  
        Circle pizza;  
        pizza = new Circle();           // Circle 객체 생성  
        pizza.radius = 10;              // 피자의 반지름을 10으로 설정  
        pizza.name = "자바피자";        // 피자의 이름 설정  
        double area = pizza.getArea();   // 피자의 면적 알아내기  
        System.out.println(pizza.name + "의 면적은 " + area);  
  
        Circle donut = new Circle();    // Circle 객체 생성  
        donut.radius = 2;               // 도넛의 반지름을 2로 설정  
        donut.name = "자바도넛";         // 도넛의 이름 설정  
        area = donut.getArea();          // 도넛의 면적 알아내기  
        System.out.println(donut.name + "의 면적은 " + area);  
    }  
}
```



자바피자의 면적은 314.0
자바도넛의 면적은 12.56

예제 4-2 : Rectangle 클래스 만들기 연습

너비와 높이를 입력 받아 사각형의 합을 출력하는 프로그램을 작성하라. 너비(width)와 높이(height) 필드, 그리고 면적 값을 제공하는 getArea() 메소드를 가진 Rectangle 클래스를 만들어 활용하라.

```
import java.util.Scanner;

public class Rectangle {
    int width;
    int height;

    public int getArea() {
        return width*height;
    }

    public static void main(String[] args) {
        Rectangle rect = new Rectangle(); // 객체 생성
        Scanner scanner = new Scanner(System.in);
        System.out.print(">> ");

        rect.width = scanner.nextInt();
        rect.height = scanner.nextInt();

        System.out.println("사각형의 면적은 " + rect.getArea());

        scanner.close();
    }
}
```

```
>> 4 5
사각형의 면적은 20
```

생성자 개념

- 생성자
 - 객체가 생성될 때 초기화를 위해 실행되는 메소드



기본 객체



생성자



초기화된 객체

생성자의 특징

- 생성자의 특징
 - 생성자는 메소드
 - 생성자 이름은 클래스 이름과 반드시 동일
 - 생성자 여러 개 작성 가능 (오버로딩)
 - 생성자는 new를 통해 객체를 생성할 때, 객체당 한 번 호출
 - 생성자는 리턴 타입을 지정할 수 없음
 - 생성자의 목적은 객체 초기화
 - 생성자는 객체가 생성될 때 반드시 호출됨.
 - 그러므로 하나 이상 선언되어야 함
 - 개발자가 생성자를 작성하지 않았으면 컴파일러가 자동으로 기본 생성자 삽입

예제 4-4 : 생성자 선언 및 활용 연습

제목과 저자를 나타내는 title과 author 필드를 가진 Book 클래스를 작성하고, 생성자를 작성하여 필드를 초기화하라.

```
public class Book {  
    String title;  
    String author;  
  
    public Book(String t) { // 생성자  
        title = t; author = "작자미상";  
    }  
  
    public Book(String t, String a) { // 생성자  
        title = t; author = a;  
    }  
  
    public static void main(String [] args) {  
        Book littlePrince = new Book("어린왕자", "생텍쥐페리");  
        Book loveStory = new Book("춘향전");  
        System.out.println(littlePrince.title + " " + littlePrince.author);  
        System.out.println(loveStory.title + " " + loveStory.author);  
    }  
}
```

어린왕자 생텍쥐페리
춘향전 작자미상

기본 생성자

- 기본 생성자(default constructor)
 - 매개 변수 없고 아무 작업 없이 단순 리턴하는 생성자
 - 디폴트 생성자라고도 부름
- 클래스에 생성자가 하나도 선언되지 않은 경우, 컴파일러에 의해 자동으로 삽입

```
public class Circle {  
    int radius;  
    void set(int r) { radius = r; }  
    double getArea() { return 3.14*radius*radius; }  
  
    public static void main(String [] args){  
        Circle pizza = new Circle();  
        pizza.set(3);  
    }  
}
```

개발자가 작성한 코드
이 코드에는 생성자가 없지만
컴파일 오류가 생기지 않음

이유

```
public class Circle {  
    int radius;  
    void set(int r) { radius = r; }  
    double getArea() { return 3.14*radius*radius; }  
  
    public Circle() {}  
  
    public static void main(String [] args){  
        Circle pizza = new Circle();  
        pizza.set(3);  
    }  
}
```

컴파일러에 의해
자동 삽입된 기본 생
성자

컴파일러가 자동으로 기본 생성자 삽입

기본 생성자가 자동 생성되지 않는 경우

- 개발자가 클래스에 생성자가 하나라도 작성한 경우
 - 기본 생성자 자동 삽입되지 않음

```
public class Circle {  
    int radius;  
    void set(int r) { radius = r; }  
    double getArea() { return 3.14*radius*radius; }  
  
    public Circle(int r) {  
        radius = r;  
    }  
    public static void main(String [] args){  
        Circle pizza = new Circle(10);  
        System.out.println(pizza.getArea());  
  
        Circle donut = new Circle();  
        System.out.println(donut.getArea());  
    }  
}
```

컴파일러가 기본 생성자를 자동 생성하지 않음

public Circle() {}

오류

컴파일 오류.
해당하는 생성자가 없음 !!!

this 레퍼런스

- this
 - 객체 자신에 대한 레퍼런스
 - 컴파일러에 의해 자동 관리, 개발자는 사용하기만 하면 됨
 - **this.멤버** 형태로 멤버 사용

```
public class Circle {  
    int radius;  
  
    public Circle() { radius = 1; }  
    public Circle(int r) { radius = r; }  
    double getArea() {  
        return 3.14*radius*radius;  
    }  
    ...  
}
```

=

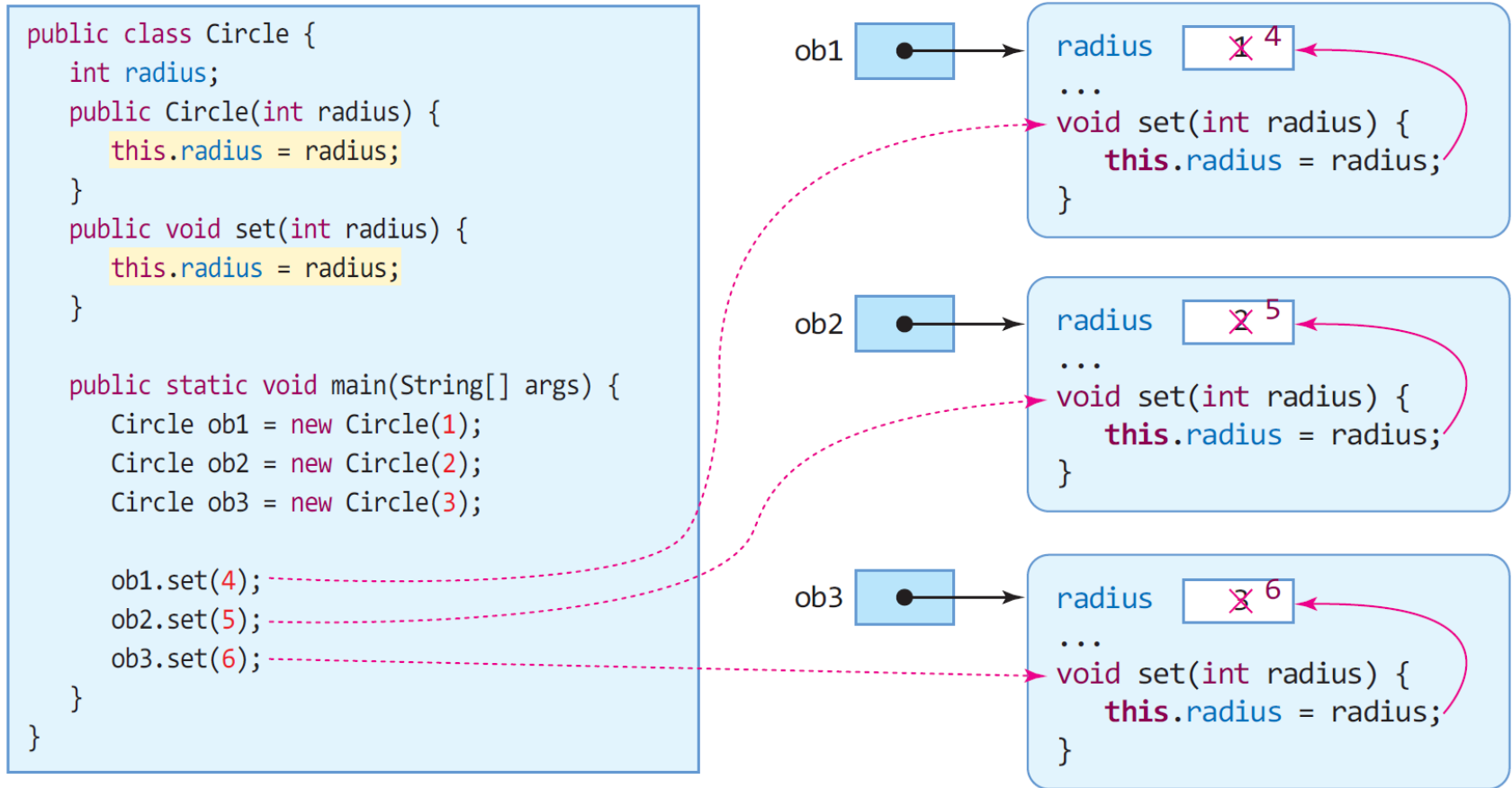
```
public class Circle {  
    int radius;  
  
    public Circle() { this.radius = 1; }  
    public Circle(int radius) {  
        this.radius = radius;  
    }  
    double getArea() {  
        return 3.14*this.radius*this.radius;  
    }  
    ...  
}
```

this를 사용하여 수정한 경우

this가 필요한 경우

- this의 필요성
 - 객체의 멤버 변수와 메소드 변수의 이름이 같은 경우
 - 다른 메소드 호출 시 객체 자신의 레퍼런스를 전달할 때
 - 메소드가 객체 자신의 레퍼런스를 반환할 때

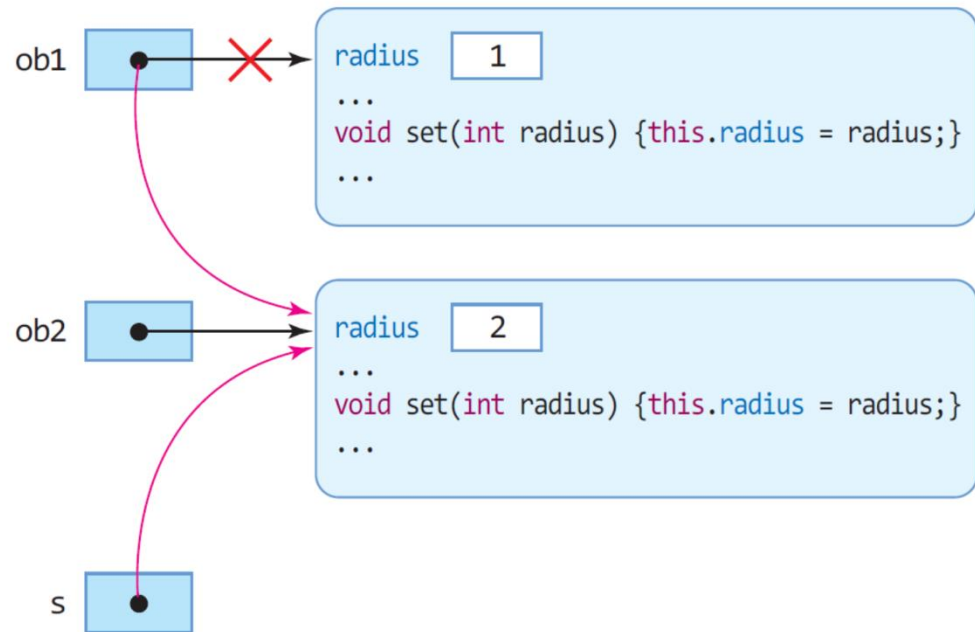
객체 속에서의 this



객체의 치환

* 객체의 치환은 객체가 복사되는 것이 아니며 레퍼런스가 복사된다.

```
public class Circle {  
    int radius;  
    public Circle(int radius) { this.radius = radius; }  
    public void set(int radius) { this.radius = radius; }  
    public static void main(String [] args) {  
        Circle ob1 = new Circle(1);  
        Circle ob2 = new Circle(2);  
        Circle s;  
  
        s = ob2;  
        ob1 = ob2; // 객체 치환  
        System.out.println("ob1.radius=" + ob1.radius);  
        System.out.println("ob2.radius=" + ob2.radius);  
    }  
}
```



ob1.radius=2
ob2.radius=2

객체 배열

- 객체 배열 생성 및 사용

```
Circle [] c;
c = new Circle[5];
for(int i=0; i<c.length; i++) // c.length는 배열 c의 크기로서 5
    c[i] = new Circle(i);
```

Circle 배열에 대한 레퍼런스 변수 c 선언

레퍼런스 배열 생성

배열의 각 원소 객체 생성

```
for(int i=0; i<c.length; i++) // 배열에 있는 모든 Circle 객체의 면적 출력
    System.out.print((int)(c[i].getArea()) + " ");
```

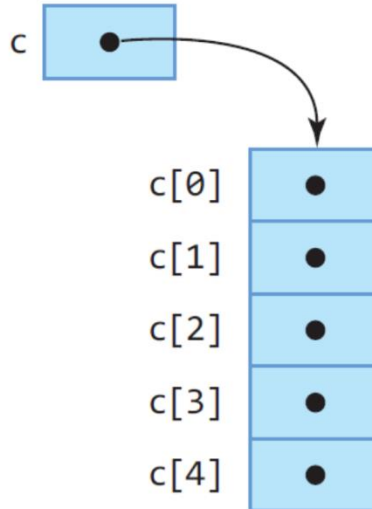
배열의 원소 객체 사용

객체 배열 선언과 생성 과정

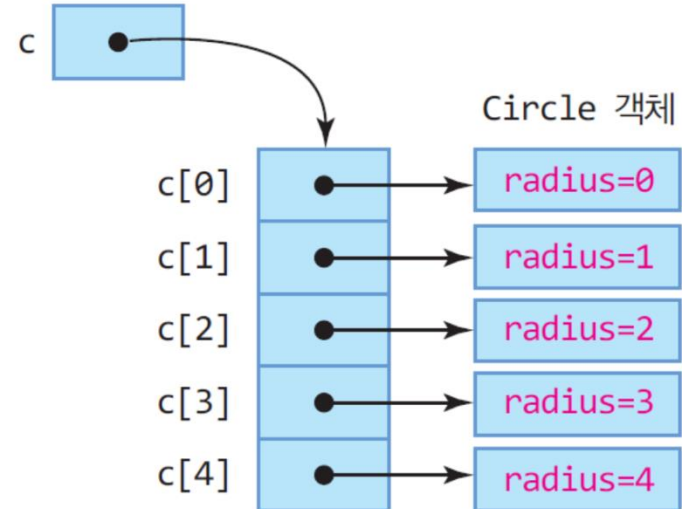
```
Circle[] c;
```



```
c = new Circle[5];
```



```
for(int i=0; i<c.length; i++)  
    c[i] = new Circle(i);
```



예제 4-6 : Circle 객체 배열 만들기

반지름이 0~4인 Circle 객체 5개를 가지는 배열을 생성하고, 배열에 있는 모든 Circle 객체의 면적을 출력하라.

```
class Circle {
    int radius;
    public Circle(int radius) {
        this.radius = radius;
    }
    public double getArea() {
        return 3.14*radius*radius;
    }
}

public class CircleArray {
    public static void main(String[] args) {
        Circle [] c;
        c = new Circle[5];

        for(int i=0; i<c.length; i++)
            c[i] = new Circle(i);

        for(int i=0; i<c.length; i++)
            System.out.print((int)(c[i].getArea()) + " ");
    }
}
```

예제 4-7 : 객체 배열 만들기 연습

예제 4-4의 Book 클래스를 활용하여
2개짜리 Book 객체 배열을 만들고,
사용자로부터 책의 제목과 저자를
입력 받아 배열을 완성하라.

제목>>사랑의 기술
저자>>에리히 프롬
제목>>시간의 역사
저자>>스티븐 호킹
(사랑의 기술, 에리히 프롬)(시간의 역사, 스티븐 호킹)

```
import java.util.Scanner;
class Book {
    String title, author;
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }
}

public class BookArray {
    public static void main(String[] args) {
        Book [] book = new Book[2]; // Book 배열 선언

        Scanner scanner = new Scanner(System.in);
        for(int i=0; i<book.length; i++) {
            System.out.print("제목>>");
            String title = scanner.nextLine();
            System.out.print("저자>>");
            String author = scanner.nextLine();
            book[i] = new Book(title, author); // 배열 원소 객체 생성
        }

        for(int i=0; i<book.length; i++)
            System.out.print("(" + book[i].title + ", " + book[i].author + ")");

        scanner.close();
    }
}
```

인자 전달

- 자바의 인자 전달 방식
 - 경우 1. 기본 타입의 값 전달
 - 값이 복사되어 전달
 - 메소드의 매개변수가 변경되어도 호출한 실인자 값은 변경되지 않음
 - 경우 2. 객체 혹은 배열 전달
 - 객체나 배열의 레퍼런스만 전달
 - 객체 혹은 배열이 통째로 복사되어 전달되는 것이 아님
 - 메소드의 매개변수와 호출한 실인자 객체나 배열 공유

인자 전달 – 기본 타입의 값이 전달되는 경우

- 매개변수가 byte, int, double 등 기본 타입의 값일 때
 - 호출자가 건네는 값이 매개변수에 복사되어 전달. 실인자 값은 변경되지 않음

→ 실행 결과

10

```
public class ValuePassing {  
    public static void main(String args[]) {  
        int n = 10;  
  
        increase(n);  
  
        System.out.println(n);  
    }  
}
```

호출

```
static void increase(int m) {  
    m = m + 1;  
}
```

main() 실행 시작

int n = 10;

n 10

increase(n);

n 10

n 10

System.out.println(n);

n 10

값 복사

increase(int m) 실행 시작

10 m

11 m

m = m + 1;

increase(int m) 종료

인자 전달 – 객체가 전달되는 경우

- 객체의 레퍼런스만 전달
 - 매개 변수가 실인자 객체 공유

→ 실행 결과

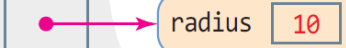
11

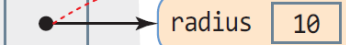
```
public class ReferencePassing {  
    public static void main (String args[]) {  
        Circle pizza = new Circle(10);  
  
        increase(pizza);  
  
        System.out.println(pizza.radius);  
    }  
}
```

호출

```
static void increase(Circle m) {  
    m.radius++;  
}  
}
```

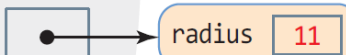

main() 실행 시작

pizza = new Circle(10); pizza 

increase(pizza); pizza 

레퍼런스 복사

increase(Circle m) 실행 시작

pizza  m 

m.radius++;

increase(Circle m) 종료

System.out.println(pizza.radius);

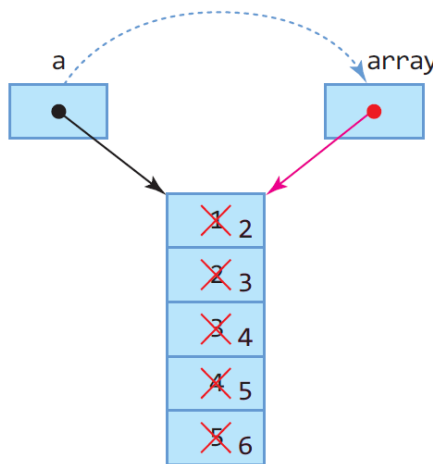
pizza 

인자 전달 - 배열이 전달되는 경우

- 배열 레퍼런스만 매개 변수에 전달
 - 배열 통째로 전달되지 않음
 - 객체가 전달되는 경우와 동일
 - 매개변수가 실인자의 배열을 공유

```
public class ArrayPassing {  
    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]+" ");  
    }  
}
```

레퍼런스 복사



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

⇒ 실행 결과

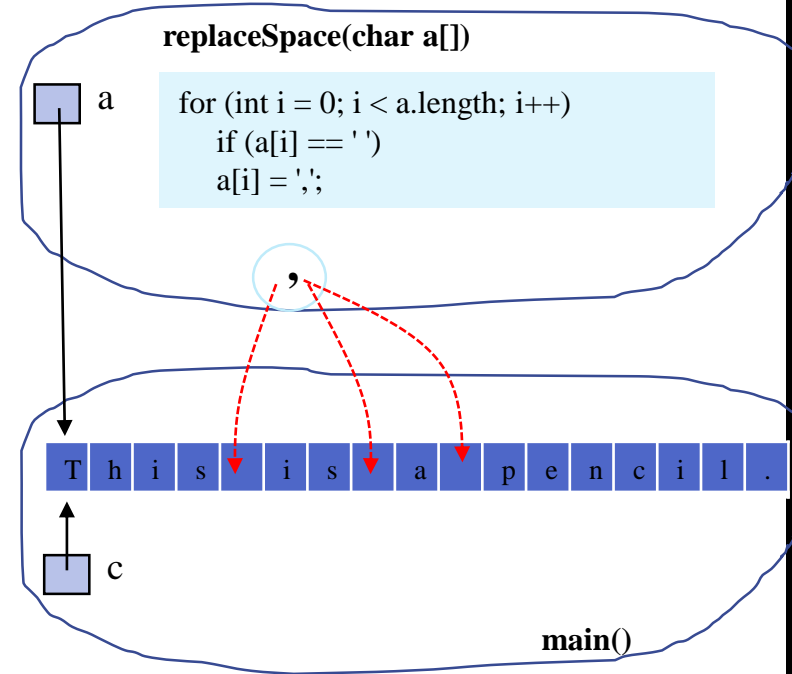
2 3 4 5 6

예제 4-8 : 인자로 배열이 전달되는 예

char[] 배열을 전달받아 출력하는 printCharArray() 메소드와 배열 속의 공백(' ') 문자를 ','로 대체하는 replaceSpace() 메소드를 작성하라.

```
public class ArrayParameterEx {  
    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.



Practice 5

- Have your own time to understand call-by-value and call-by-reference

메소드 오버로딩

- 메소드 오버로딩(Overloading)
 - 이름이 같은 메소드 작성
 - 매개변수의 개수나 타입이 서로 다르고
 - 이름이 동일한 메소드들
 - 리턴 타입은 오버로딩과 관련 없음

// 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;  
    }  
}
```

// 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);  
    }  
}
```

두 개의 getSum() 메소드는 매
개변수의 개수, 타입이 모두 같
기 때문에 메소드 오버로딩 실패

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);  
}
```

매개 변수의 개수와 타입이
서로 다른 3 함수 호출

```
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;  
    }  
}
```

Practice 6

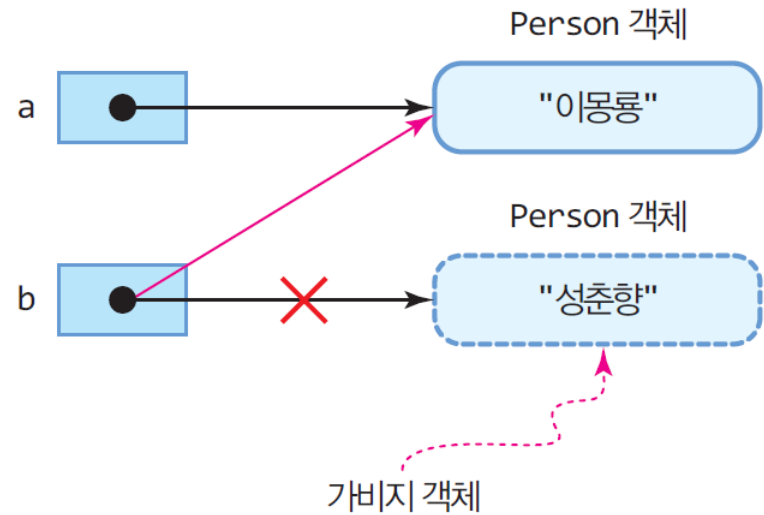
- Provides overloading methods
 - `push(String a, String b);`
 - `push(String a, String b, String c);`

객체의 소멸과 가비지 컬렉션

- 객체 소멸
 - `new`에 의해 할당된 객체 메모리를 자바 가상 기계의 가용 메모리로 되돌려 주는 행위
- 자바 응용프로그램에서 임의로 객체 소멸할 수 없음
 - 객체 소멸은 자바 가상 기계의 고유한 역할
 - 자바 개발자에게는 매우 다행스러운 기능
 - C/C++에서는 할당받은 객체를 개발자가 되돌려 주어야 함
 - C/C++ 프로그램 작성을 어렵게 만드는 요인
- 가비지
 - 가리키는 레퍼런스가 하나도 없는 객체
 - 누구도 사용할 수 없게 된 메모리
- 가비지 컬렉션
 - 자바 가상 기계의 가비지 컬렉터가 자동으로 가비지 수집 반환

가비지 사례

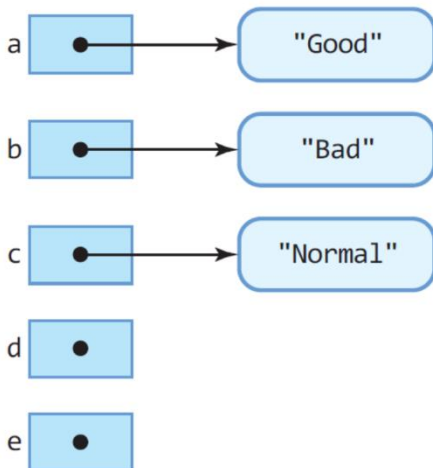
```
Person a, b;  
a = new Person("이몽룡");  
b = new Person("성춘향");  
b = a; // b가 가리키던 객체는 가비지가 됨
```



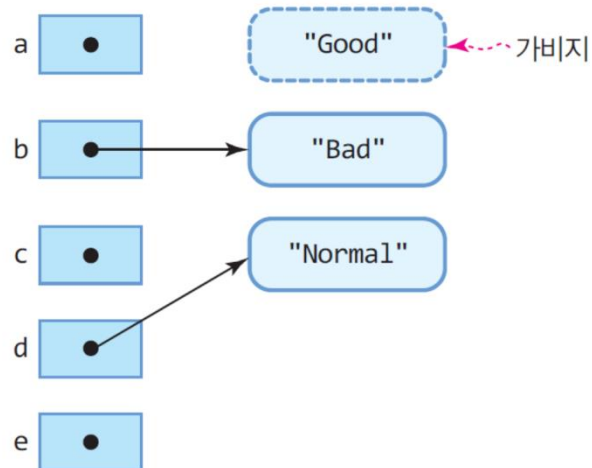
예제 4-9 : 가비지의 발생

다음 코드에서 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
 - 자바에서 가비지를 자동 회수하는 과정
 - 가용 메모리로 반환
 - 가비지 컬렉션 스레드에 의해 수행
- 개발자에 의한 강제 가비지 컬렉션
 - System 또는 Runtime 객체의 gc() 메소드 호출
- 이 코드는 자바 가상 기계에 강력한 가비지 컬렉션 요청
 - 그러나 자바 가상 기계가 가비지 컬렉션 시점을 전적으로 판단

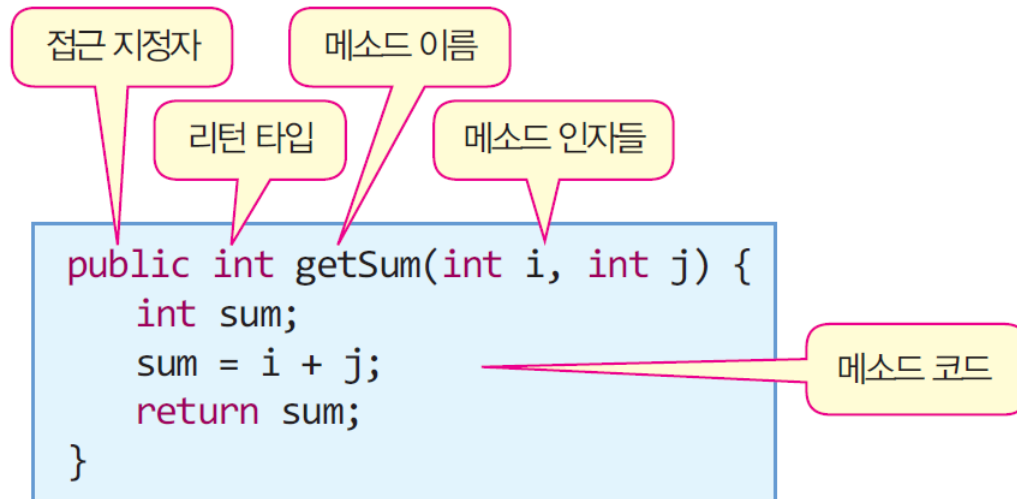
```
System.gc(); // 가비지 컬렉션 작동 요청
```

Practice 7

- Learn how to use JConsole

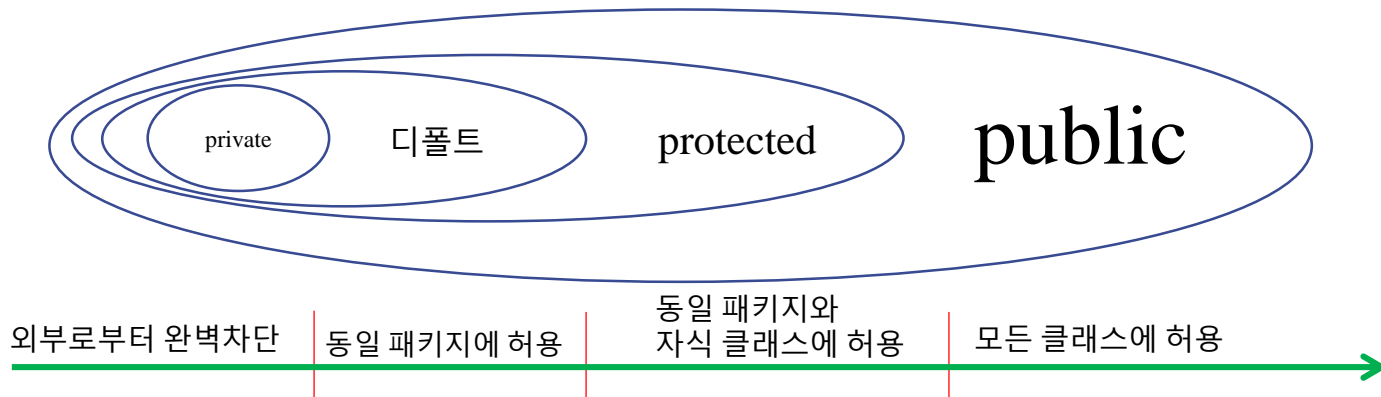
메소드 형식

- 메소드
 - 클래스의 멤버 함수, C/C++의 함수와 동일
 - 자바의 모든 메소드는 반드시 클래스 안에 있어야 함(캡슐화 원칙)
- 메소드 구성 형식
 - 접근 지정자
 - public, private, protected, 디폴트(접근 지정자 생략된 경우)
 - 리턴 타입
 - 메소드가 반환하는 값의 데이터 타입



접근 지정자

- 자바의 접근 지정자
 - 4가지
 - private, protected, public, 디폴트(접근지정자 생략)
- 접근 지정자의 목적
 - 클래스나 일부 멤버를 공개하여 다른 클래스에서 접근하도록 허용
 - 객체 지향 언어의 캡슐화 정책은 멤버를 보호하는 것
 - 접근 지정은 캡슐화에 묶인 보호를 일부 해제할 목적
- 접근 지정자에 따른 클래스나 멤버의 공개 범위

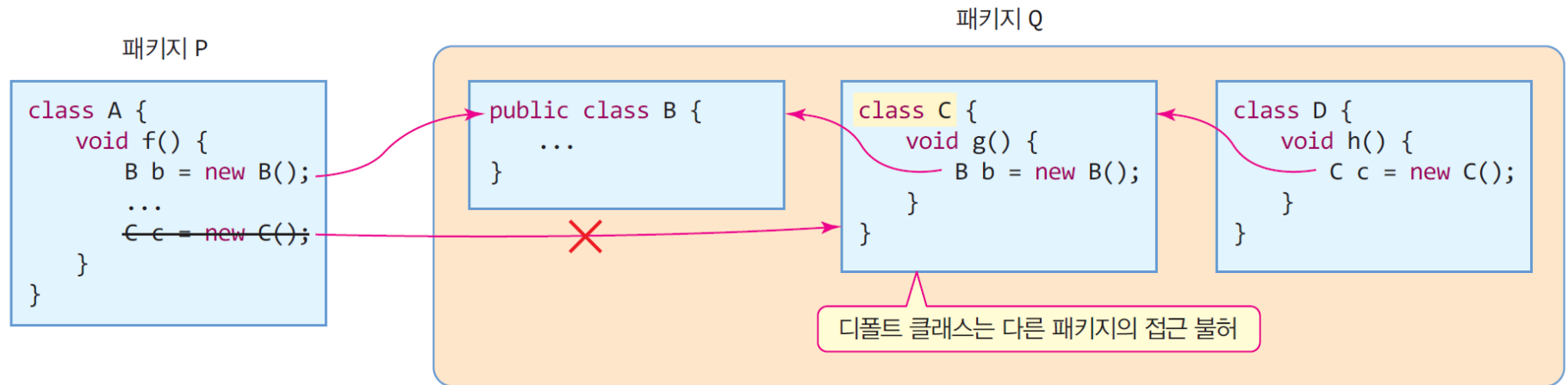


클래스 접근 지정

- 클래스 접근지정
 - 다른 클래스에서 사용하도록 허용할 지 지정
 - public 클래스
 - 다른 모든 클래스에게 접근 허용
 - 디폴트 클래스(접근지정자 생략)
 - package-private라고도 함
 - 같은 패키지의 클래스에만 접근 허용

```
public class World { // public 클래스
.....
}
```

```
class Local { // 디폴트 클래스
.....
}
```



public 클래스와 디폴트 클래스의 접근 사례

멤버 접근 지정

- public 멤버
 - 패키지에 관계 없이 모든 클래스에게 접근 허용
- private 멤버
 - 동일 클래스 내에만 접근 허용
 - 상속 받은 서브 클래스에서 접근 불가
- protected 멤버
 - 같은 패키지 내의 다른 모든 클래스에게 접근 허용
 - 상속 받은 서브 클래스는 다른 패키지에 있어도 접근 가능
- 디폴트(default) 멤버
 - 같은 패키지 내의 다른 클래스에게 접근 허용

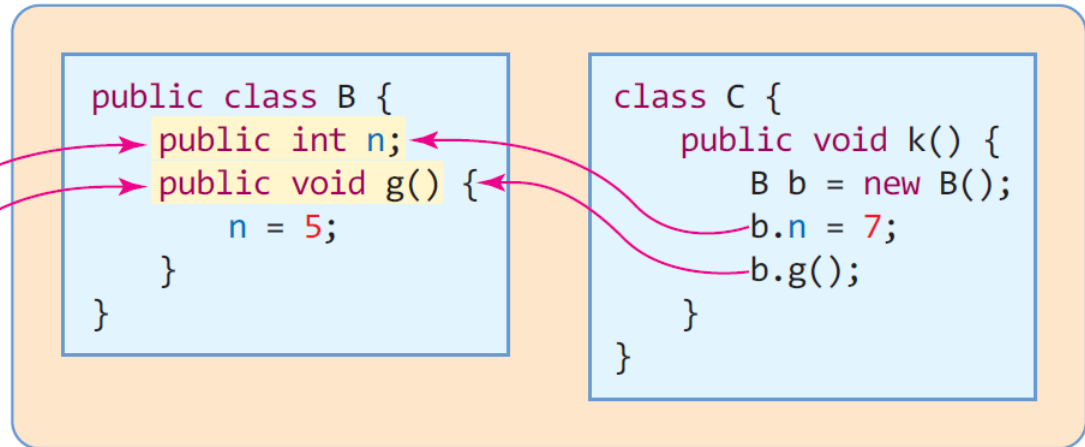
멤버에 접근하는 클래스	멤버의 접근 지정자			
	private	디폴트 접근 지정	protected	public
같은 패키지의 클래스	×	○	○	○
다른 패키지의 클래스	×	×	×	○
접근 가능 영역	클래스 내	동일 패키지 내	동일 패키지과 자식 클래스	모든 클래스

멤버 접근 지정자의 이해

public 접근 지정 사례

패키지 P

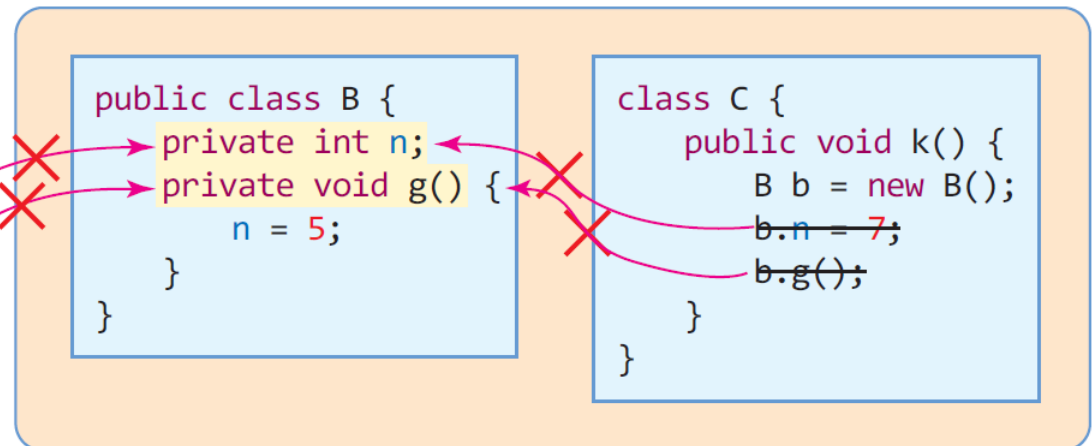
```
class A {  
    void f() {  
        B b = new B();  
        b.n = 3;  
        b.g();  
    }  
}
```



private 접근 지정 사례

패키지 P

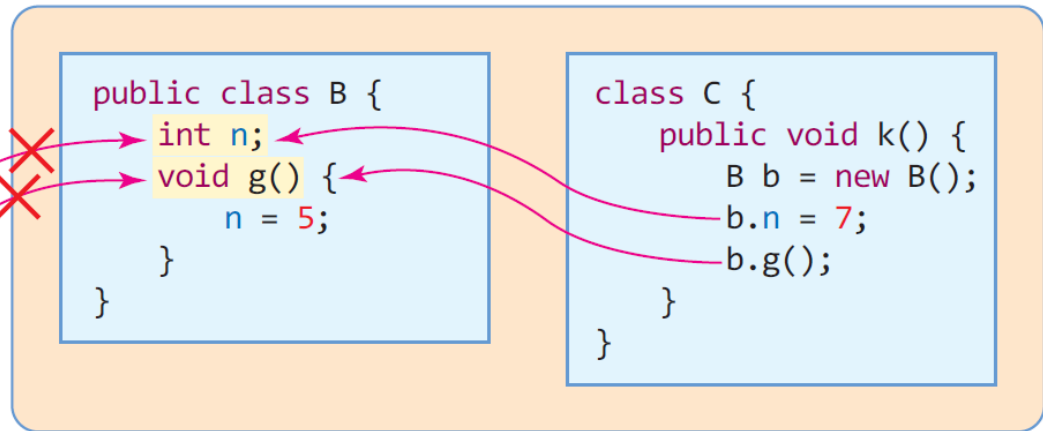
```
class A {  
    void f() {  
        B b = new B();  
        b.n = 3;  
        b.g();  
    }  
}
```



디폴트 접근 지정 사례

패키지 P

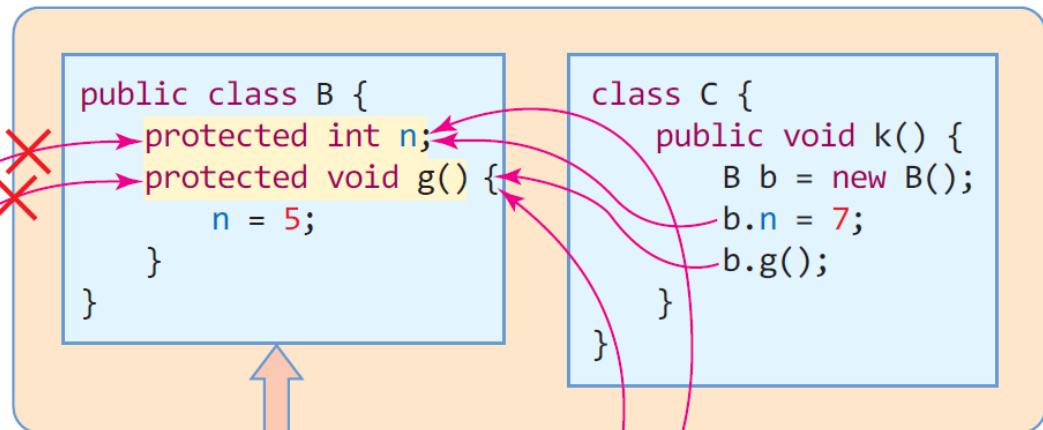
```
class A {
    void f() {
        B b = new B();
        b.n = 3;
        b.g();
    }
}
```



protected 접근 지정 사례

패키지 P

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



D가 B를 상속받음

extends는 상속받음을 나타냄

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

예제 4-10 : 멤버의 접근 지정자

다음 코드의 두 클래스 Sample과 AccessEx 클래스는 동일한 패키지에 저장된다.
컴파일 오류를 찾아 내고 이유를 설명하라.

```
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)
```

Practice 8

- Make your data private and provide **getter and setter**
- Utilize the constructor in your program

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.

non-static 멤버와 static 멤버의 차이

	non-static 멤버	static 멤버
선언	<pre>class Sample { int n; void g() {...} }</pre>	<pre>class Sample { static int m; static void g() {...} }</pre>
공간적 특성	멤버는 객체마다 별도 존재 • 인스턴스 멤버라고 부름	멤버는 클래스당 하나 생성 • 멤버는 객체 내부가 아닌 별도의 공간(클래스 코드가 적재되는 메모리)에 생성 • 클래스 멤버라고 부름
시간적 특성	객체 생성 시에 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	공유되지 않음 • 멤버는 객체 내에 각각 공간 유지	동일한 클래스의 모든 객체들에 의해 공유됨

static의 활용

1. 전역 변수와 전역 함수를 만들 때 활용

- 전역변수나 전역 함수는 static으로 클래스에 작성
- static 멤버를 가진 클래스 사례
 - Math 클래스 : java.lang.Math
 - 모든 필드와 메소드가 public static으로 선언
 - 다른 모든 클래스에서 사용할 수 있음

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

// 잘못된 사용법

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

// 바른 사용법

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

2. 공유 멤버를 작성할 때

- static 필드나 메소드는 하나만 생성. 클래스의 객체들 공유

static 메소드의 제약 조건 1

- static 메소드는 non-static 멤버 접근할 수 없음
 - 객체가 생성되지 않은 상황에서도 static 메소드는 실행될 수 있기 때문에, non-static 메소드와 필드 사용 불가
 - 반대로, non-static 메소드는 static 멤버 사용 가능

```
class StaticMethod {  
    int n;  
    void f1(int x) {n = x;} // 정상  
    void f2(int x) {m = x;} // 정상
```

```
    static int m;
```

오류

```
    static void s1(int x) {n = x;} // 컴파일 오류. static 메소드는 non-static 필드  
                                    사용 불가
```

오류

```
    static void s2(int x) {f1(3);} // 컴파일 오류. static 메소드는 non-static 메소드  
                                    사용 불가
```

```
    static void s3(int x) {m = x;} // 정상. static 메소드는 static 필드 사용 가능
```

```
    static void s4(int x) {s3(3);} // 정상. static 메소드는 static 메소드 호출 가능
```

```
}
```

static 메소드의 제약 조건 2

- static 메소드는 this 사용불가
 - static 메소드는 객체가 생성되지 않은 상황에서도 호출이 가능하므로, 현재 객체를 가리키는 this 레퍼런스 사용할 수 없음

```
class StaticAndThis {  
    int n;  
    static int m;  
    void f1(int x) {this.n = x;}  
    void f2(int x) {this.m = x;} // non-static 메소드에서는 static 멤버 접근 가능  
    static void s1(int x) {this.n = x;} // 컴파일 오류. static 메소드는 this 사용 불가  
    static void s2(int x) {this.m = x;} // 컴파일 오류. static 메소드는 this 사용 불가  
}
```

오류

오류

Table of Contents

- Class
 - Class / Instance
 - Stack
 - Constructor
 - Call-by-value / Call-by-reference
 - Method overloading
 - Garbage collection
 - Access modifier
 - Static variables / methods