

The background image shows a dark, narrow street at night. A broom with a wooden handle and straw bristles leans against a brick wall on the right. A pile of dry, brown leaves is on the ground near the broom. In the distance, streetlights and a trash can are visible. The overall mood is quiet and somewhat somber.

경기과학기술대학교 · 인공지능학과

자바 프로그래밍

7주차

가비지 컬렉션과 접근 지정자, static

쓰지 않는 객체는 치우고,
감출 것은 감추고 함께 쓸 것은 함께 쓴다.

오늘의 강의

1 교시 가비지와 클래스 접근, 2 교시 멤버의 접근 지정자, 3 교시 static 과 final

1교시



가비지와 가비지 컬렉션

아무도 가리키지 않는 객체
클래스에 붙는 public

2교시



멤버의 접근 지정자

private 은 클래스 안에서만
getter 와 setter

3교시



static 과 final, 과제

모두가 함께 쓰는 하나
바꿀 수 없는 상수

1교시

가비지와 가비지 컬렉션

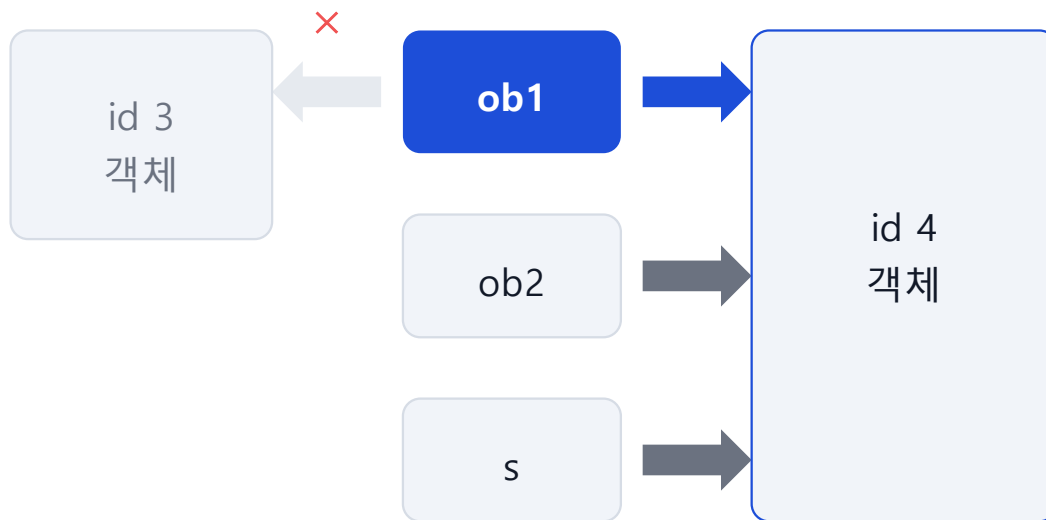
아무도 가리키지 않는 객체는 누가 치우는지 본다.

가비지

가비지 컬렉션

클래스 접근

지난주 ob1 이 가리키던 객체는 갈 곳이 없어졌다



ob1 = ob2; // 객체 치환.
ob1 은 ob2 의 객체를 가리킨다

가리키는 것이 없는 객체는 메모리에 계속 남을까

아무도 가리키지 않는 id 3 객체는 어떻게 될까.

① 프로그램이 끝날 때까지 남는다

② 누군가 치운다

자바에는 객체를 없애는 연산자가 없다

자바에서는 객체를 생성하는 new 연산자는 있지만 객체를 소멸시키는 연산자는 없다.

그러므로 개발자가 마음대로 객체를 소멸시킬 수 없다.

만드는 말: new

없애는 말: 없음

C/C++ 은 개발자가 돌려주고 자바는 JVM 이 거둔다

C/C++

다 쓴 메모리를 개발자가 직접 돌려준다
안 돌려주면 언젠가 메모리가 모자란다

자바

쓰지 않는 객체를 자바 가상 기계가 거둔다
돌려주는 코드를 쓰지 않는다

그래서 자바 코드에는 지우는 줄이 없다.

가비지

할당받은 객체나 배열 메모리 중에서 더 이상 사용하지 않게 된 메모리를 가비지(garbage)라고 부른다.

garbage

참조하는 레퍼런스가 하나도 없으면 가비지다

자바 플랫폼은 참조하는 레퍼런스가 하나도 없는 객체나 배열을 가비지로 판단한다.

왜냐하면 이 객체는 프로그램에서 더 이상 접근할 수 없게 되었기 때문이다.

레퍼런스 0개



접근할 수 없다



가비지

가비지도 거두기 전까지는 메모리를 차지한다

가비지는 다시 갈 길이 없을 뿐, 메모리에서 바로 사라지지 않는다.



 쓰는 객체

 가비지

 빈 메모리

Person 객체 둘을 만드는 클래스

Ex01_PersonGarbage.java

```
class Person {  
    String name;  
  
    Person(String name) {  
        this.name = name;  
    }  
}
```

b = a 앞뒤로 b.name 출력하기

Ex01_PersonGarbage.java

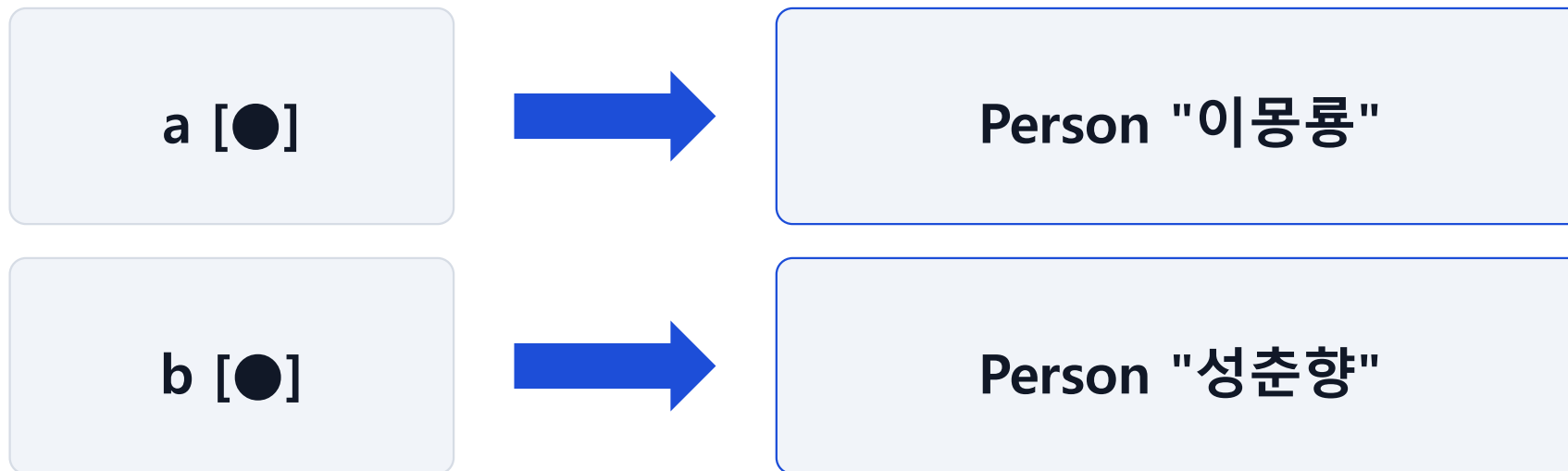
```
public class Ex01_PersonGarbage {  
    public static void main(String[] args) {  
        Person a, b;  
        a = new Person("이몽룡");  
        b = new Person("성춘향");  
        System.out.println(b.name);  
        b = a; // b가 가리키던 객체는 가비지가 됨  
        System.out.println(b.name);  
    }  
}
```

성춘향 다음에 이몽룡

성춘향
이몽룡

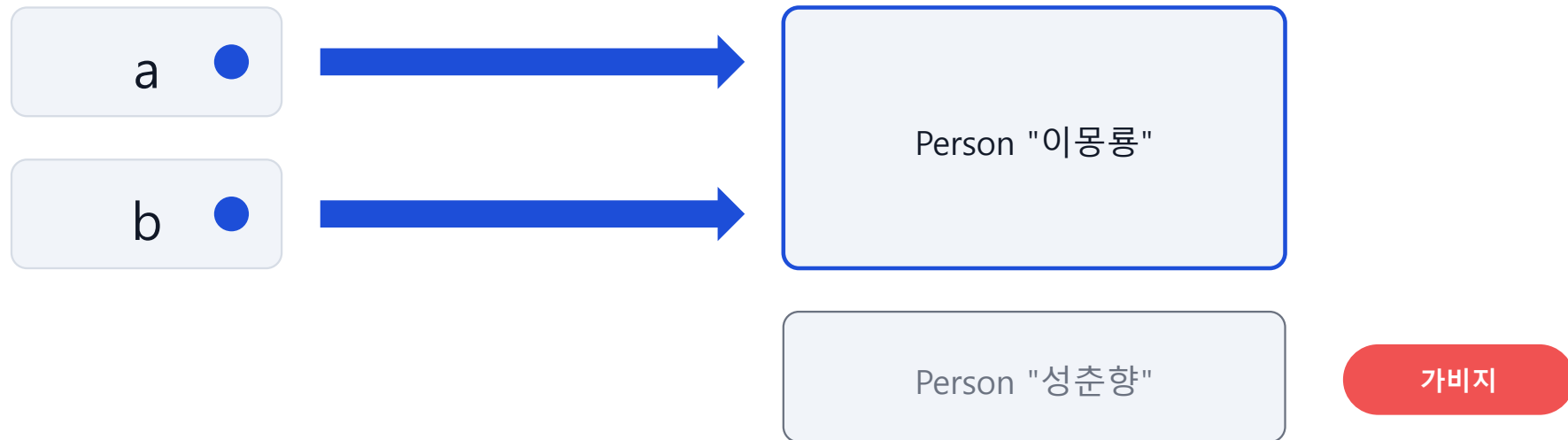
b 가 가리키는 객체가 바뀌었다

b = a 하기 전에는 객체가 둘, 레퍼런스도 둘



레퍼런스마다 객체 하나씩.

b = a 다음 성춘향 객체는 가비지가 된다



가리키는 레퍼런스가 없는 객체가 가비지다.

문제: 이 코드에서 언제 가비지가 생길까

다음 소스에서 언제 가비지가 발생하는지 설명하라.

줄마다 어느 객체를 누가 가리키는지 따라가 본다.

예제 4-9 문자열 셋과 레퍼런스 다섯

Ex02_GarbageEx.java

결과를 먼저 예측한 뒤 실행해 비교한다.

```
public class Ex02_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;  
        System.out.println(a);  
        System.out.println(d);  
        System.out.println(c);  
    }  
}
```

세 줄은 무엇을 찍을까

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

()

```
System.out.println(d);
```

()

```
System.out.println(c);
```

()

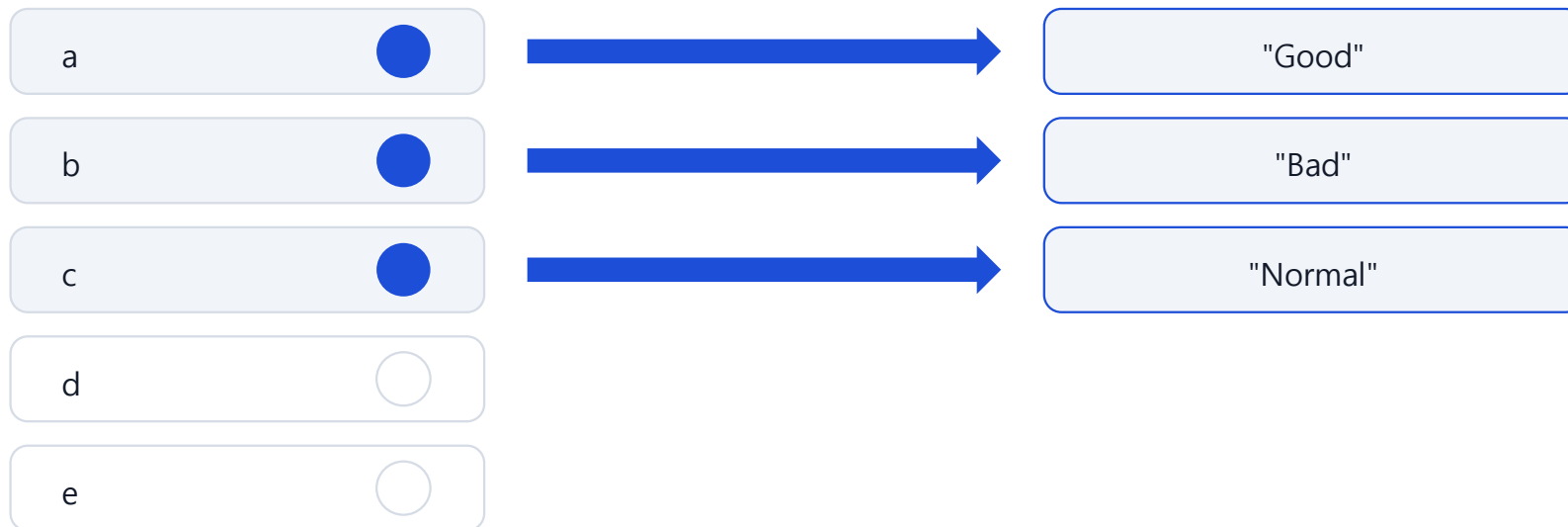
오른쪽 칸은 실행 뒤에 채운다.

— null, Normal, null

null
Normal
null

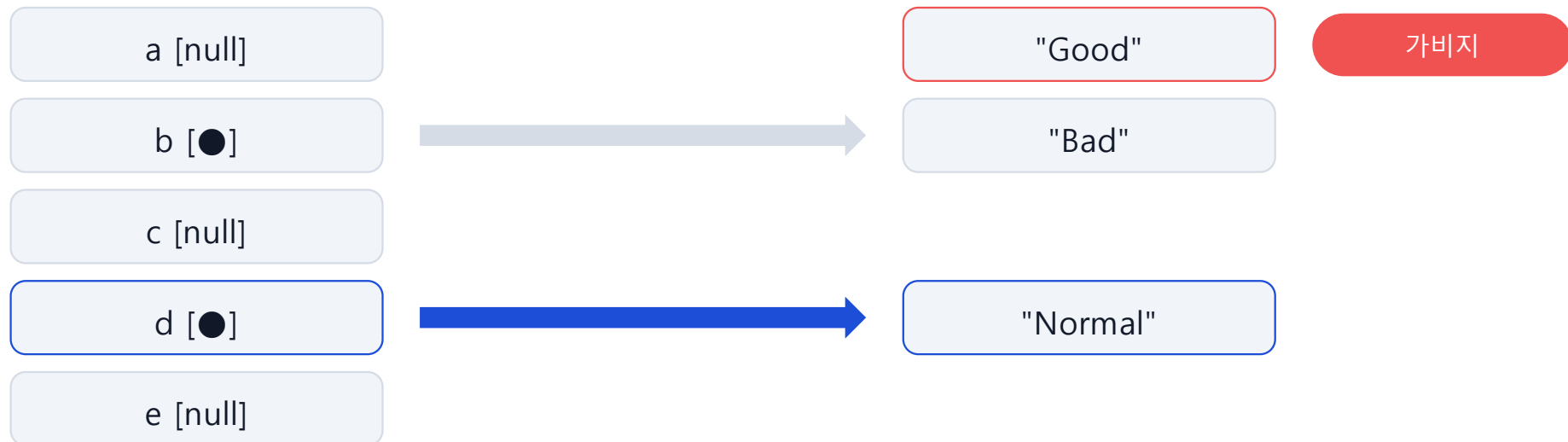
가리키는 것이 없는 레퍼런스는 null 을 찍는다

라인 6까지는 문자열 셋이 모두 살아 있다



여기까지는 가비지가 없다.

끝까지 실행하면 Good 이 가비지가 된다



c 가 null 이 돼도 d 가 Normal 을 가리키니 Normal 은 가비지가 아니다.

— null 을 넣거나 다른 객체를 가리키면 앞 객체는 가 비지가 된다

레퍼런스 = null;

레퍼런스 = 다른 레퍼런스;

둘 다 앞 객체를 가리키던 마지막 줄을 끊는다.

장난감 세 개를 만드는 Toy 클래스

Ex03_ToyGarbage.java

두 줄을 써서 공과 차 객체를 가비지로 만든다.

```
class Toy {  
    String name;  
  
    Toy(String n) {  
        name = n;  
    }  
}
```

공과 차 객체를 가비지로 만들어 보기

Ex03_ToyGarbage.java

```
public class Ex03_ToyGarbage {  
    public static void main(String[] args) {  
        Toy t1 = new Toy("곰");  
        Toy t2 = new Toy("공");  
        Toy t3 = new Toy("차");  
        // 여기서 부터 코드를 작성하세요.  
  
        // 여기 까지  
        System.out.println(t1.name + " " + t2.name);  
        System.out.println(t3 == null);  
    }  
}
```

곰 곰, true

곰 곰
true

t2 는 곰을, t3 은 아무것도 가리키지 않는다

빈칸 그대로 실행하면

곰 공
false

아무것도 끊지 않으면 객체 셋이 모두 남는다

가비지가 된 것은 공과 차

객체	가리키는 레퍼런스	가비지?
곰	t1, t2	아니다
공	없음	그렇다
차	없음	그렇다

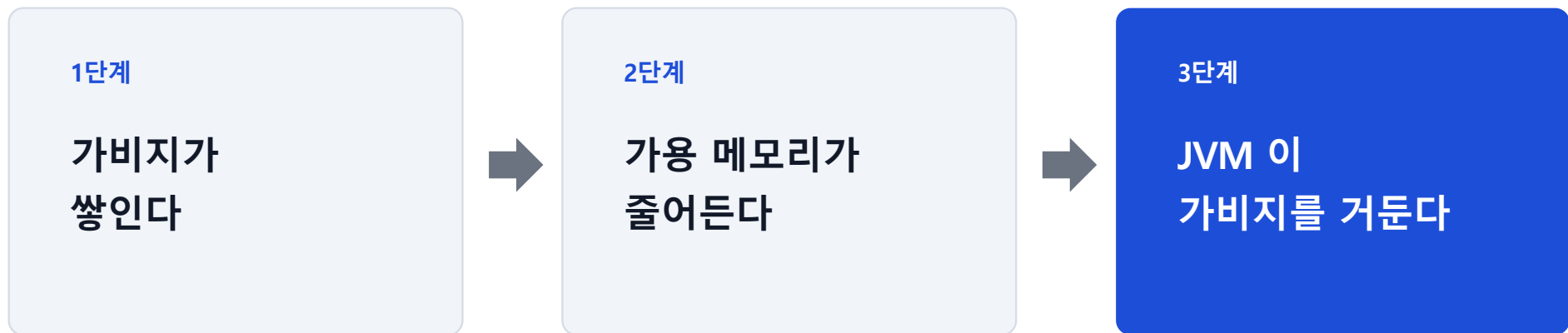
가비지 컬렉션

garbage collection

가용 메모리 공간이 일정 크기 이하로 줄어들면, 자바 가상 기계는 자동으로 가비지를 회수하여 가용 메모리 공간을 늘린다.

이것을 가비지 컬렉션이라고 부른다.

가용 메모리가 줄면 자바 가상 기계가 스스로 거둔다



프로그램 코드에는 거두는 줄이 없다.

가비지 컬렉션은 가비지 컬렉션 스레드가 맡는다

가비지 컬렉션 스레드

가비지 컬렉션은 자바 가상 기계 내에 준비된 가비지 컬렉션 스레드(thread)에 의해 처리된다.

스레드

스레드는 프로그램 안에서 따로 도는 일의 흐름이다. 자세한 것은 교재 12장.

가비지 컬렉션이 언제 일어나는지는 알 수 없다

자바 가상 기계의 판단에 의해 가비지 컬렉션이 수행되므로
언제 가비지 컬렉션이 일어나는지 알 수 없다.

정하는 쪽: JVM

프로그램: 모른다

큰 프로그램은 거두는 동안 잠깐 멈춘 것처럼 보인다

이때 가비지 컬렉션이 실행되며, 자바 프로그램은 실행을 멈추고 가비지 컬렉션이 끝나기를 기다리게 되어, 사용자의 눈에는 프로그램이 중지된 것처럼 보인다.

실행

멈춤

실행

그래서 자바는 실시간 처리에 맞지 않는다고 알려져 있다

이런 이유로 자바는 실시간 처리 목적에는
부적합한 것으로 알려져 있다.

제때 반응해야 하는 일

몇 분의 1초도 늦으면 안 되는 일



2분마다 멈췄하던 메신저

1

메신저 Discord 의 Read States 서비스는 자바가 아니라 구글이 만든 언어 Go 로 짰다. Go 도 가비지 컬렉션 언어다.

2

Go 는 적어도 2분에 한 번 가비지 컬렉션을 강제로 돌렸다("Go will force a garbage collection run every 2 minutes at minimum").

3

그때마다 메모리의 큰 캐시를 다 훑느라 응답이 크게 느려졌고, Discord 는 이 서비스를 다른 언어 Rust 로 다시 짰다.

가비지 컬렉션은 공짜가 아니다

기다림

거두는 동안 프로그램은 기다린다

불확실성

언제 거둘지는 프로그램이 정하지 못한다

그래도 자바는 쓰지 않는 객체를 개발자가 지우지 않아도 된다.

System.gc() 는 거둬 달라는 요청이다

```
System.gc(); // 가비지 컬렉션 작동 요청
```

System 이나 Runtime 의 gc() 를 부르면 가비지 컬렉션을 요청할 수 있다.

요청일 뿐, 거두는 때는 여전히 JVM 이 정한다

그러나 이 문장을 호출한 즉시 가비지 컬렉션이 작동되는 것은 아닙니다.

이 문장은 자바 가상 기계에게 가비지 컬렉션이 필요하다는 강력한 제안에 불과하다.

문제: 가비지 컬렉션이 필요한 까닭과 Scanner 반복

1 자바에서 가비지 컬렉션이 필요한 이유는 무엇인가?

2 다음은 키보드로부터 10개의 정수를 읽는 코드이다. 가비지가 발생하는지 답하라.

```
int [] n = new int [10];  
for(int i=0; i<10; i++) {  
    Scanner s = new Scanner(System.in);  
    n[i] = s.nextInt();  
}
```

풀이 1: 지우는 코드 없이도 메모리가 모자라지 않게

자바에는 객체를
없애는 연산자가
없다.



썬이는 가비지를
누군가 거둬야
가용 메모리가
0 이 되지 않는다.

풀이 2: 반복마다 앞 Scanner 가 가비지가 된다

```
for(int i=0; i<10; i++) {  
    Scanner s = new Scanner(System.in);  
    n[i] = s.nextInt();  
}
```

반복마다 새 객체

s 가 새 Scanner 를 가리키면 앞 Scanner 를 가리키는 레퍼런스가 없어진다.

지난주 메소드 맨 앞의 public

접근 지정자



```
public int getSum(int i, int j) {
```

지난주에는 이름표만 붙였다. 오늘 그 뜻을 본다.

패키지는 폴더와 같다

패키지는 디렉터리 혹은 폴더와 같은 개념이며, 일반적으로 개발자는 개발한 클래스 파일을 관련된 것끼리 묶어 여러 패키지에 분산 관리한다.



패키지 P

클래스

클래스

클래스



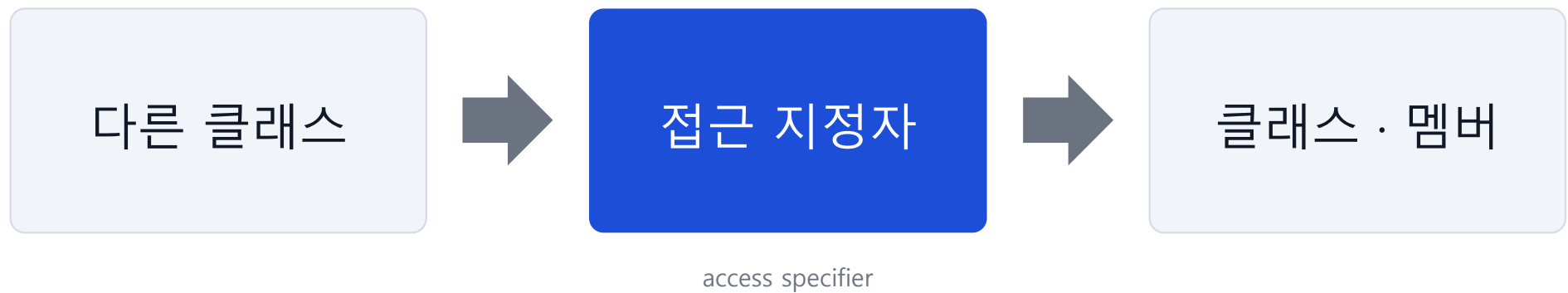
패키지 Q

클래스

클래스

접근 지정자

접근 지정자는 클래스나 멤버들을 다른 클래스에서 접근해도 되는지의 여부를 선언하는 지시어이다.



자바의 접근 지정자는 넷이다

private

protected

public

**(생략)
디폴트**

private, protected, public, 접근 지정자 생략(디폴트 접근 지정)

클래스에는 public 이나 디폴트를 붙인다

public 클래스

```
public class World { }
```

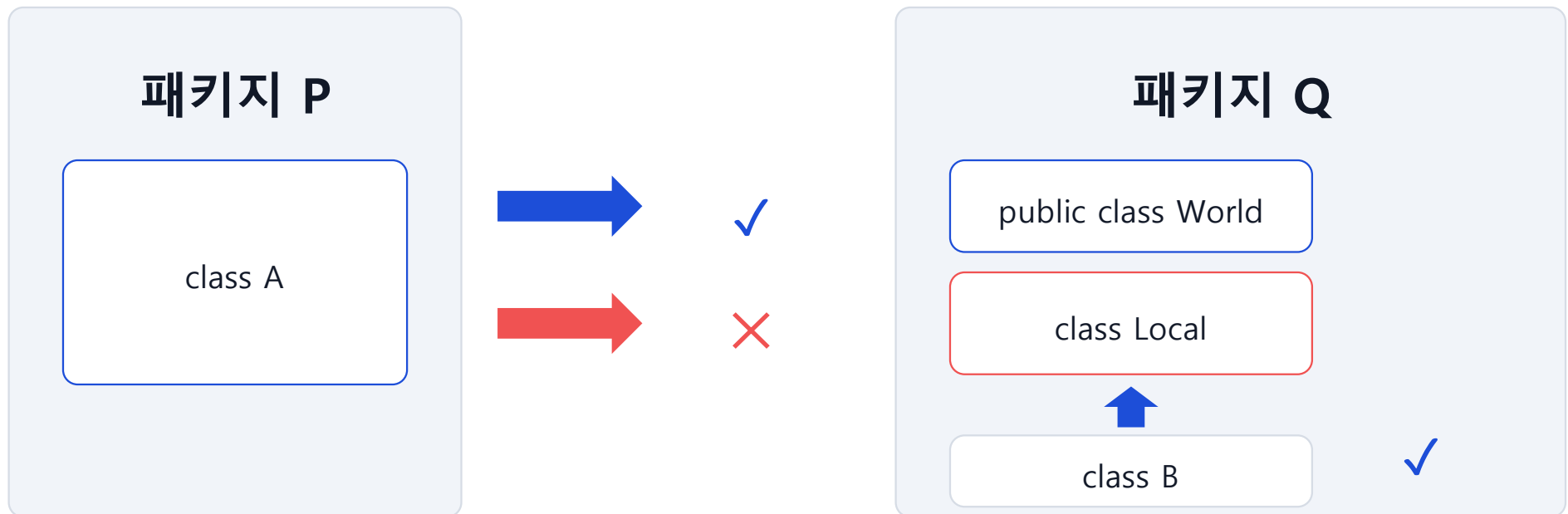
패키지에 상관없이 누구나 쓴다

디폴트 클래스

```
class Local { }
```

같은 패키지 안에서만 쓴다

디폴트 클래스는 다른 패키지에서 쓸 수 없다



실습실 파일 하나는 같은 패키지라 오늘은 ✓ 쪽만 직접 본다.

같은 파일의 디폴트 클래스 Local 쓰기

Ex04_DefaultClass.java

```
class Local { // 디폴트 클래스
    void hello() {
        System.out.println("같은 패키지에서는 Local 을 쓸 수 있다");
    }
}

public class Ex04_DefaultClass { // public 클래스
    public static void main(String[] args) {
        Local l = new Local();
        l.hello();
    }
}
```



같은 패키지라 Local 을 쓸 수 있다

같은 패키지에서는 Local 을 쓸 수 있다

같은 파일은 같은 패키지다

한 파일에 public 클래스를 둘 두면

Ex05_TwoPublic.java

```
public class Helper {  
    void hello() {  
        System.out.println("안녕");  
    }  
}  
  
public class Ex05_TwoPublic {  
    public static void main(String[] args) {  
        Helper h = new Helper();  
        h.hello();  
    }  
}
```

public 이 하나 더

컴파일 오류 원문

Ex05_TwoPublic.java:1: error: class Helper is public, should be declared in a file named Helper.java

```
public class Helper {
```

^

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

public 클래스 이름은 파일 이름과 같아야 한다

한 파일에 public 클래스는 하나,
이름은 파일 이름과 같게

public class 는
파일마다 하나다

그 이름 = 파일 이름

(Ex05_TwoPublic.java)

```
class Helper {
```

1교시에서 본 것

1

자바에는 객체를 없애는 연산자가 없다

2

가리키는 레퍼런스가 하나도 없는 객체가 가비지다

3

가비지는 JVM 이 알아서 거두고, 언제 거둘지는 알 수 없다

4

클래스에는 public 이나 디폴트를 붙인다

1교시 실습 살펴보기

실습 ①

빈 객체가 가비지로

t2 = t1 을 하면 빈 객체는 가리키는 것이 없어진다.

가비지는 줄 하나로 생긴다.

Ex05 오류 예제

public 클래스 둘

public 클래스를 둘 두면 실행 전에 컴파일에서 막힌다.

2교시

멤버의 접근 지정자

필드와 메소드를 누구에게 열고 누구에게 닫을지 정한다.

표 4-1

private

getter · setter

공개 범위는

private < 디폴트 < protected < public 순으로 넓어진다

멤버에 대한 공개의 범위는 private < 디폴트 < protected < public 순으로 넓어진다.

private

클래스 안

디폴트

같은 패키지

protected

패키지 · 자식

public

모든 클래스

표 4-1 접근 지정자에 따른 멤버 접근 레벨

멤버에 접근하는 클래스	private	디폴트 접근 지정	protected	public
같은 패키지의 클래스	X	O	O	O
다른 패키지의 클래스	X	X	X	O
접근 가능 영역	클래스 내	동일 패키지 내	동일 패키지와 자식 클래스	모든 클래스

오늘 실행으로 확인하는 줄은 「같은 패키지」 줄

	private	디폴트	protected	public
같은 패키지의 클래스	X	O	O	O
다른 패키지의 클래스	X	X	X	O

실습실에서 실행

그림으로만

실습실은 파일 하나만 컴파일한다. 파일 하나는 패키지 하나다.

public 멤버는 패키지를 가리지 않고 모든 클래스가 접근한다

public 멤버는 public으로 선언된 것으로, 패키지를 막론하고 모든 클래스들이 접근 가능하다.

같은 패키지의 클래스

다른 패키지의 클래스

다른 패키지의 A 도 같은 패키지의 C 도 B 의 public 멤버를 쓴다



private 멤버는 그 클래스 안에서만 쓴다

private 접근 지정자는 비공개를 의미하는 것으로 private 멤버는 클래스 내의 멤버들에게만 접근이 허용된다.

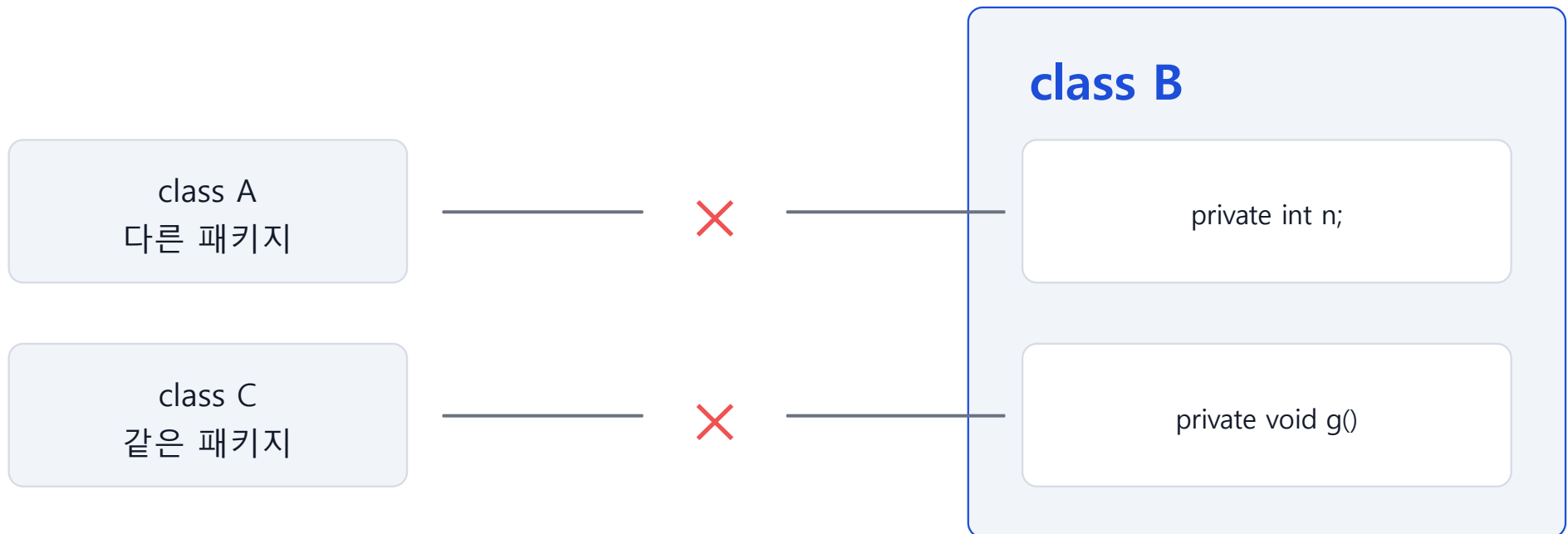
열린 곳

같은 클래스 안의 멤버

닫힌 곳

같은 패키지의 다른 클래스도 쓰지 못한다

A 도 C 도 B 의 private 멤버는 쓰지 못한다



protected 멤버는 같은 패키지와 자식 클래스에 열린다

protected 접근 지정자는 보호된 공개를 의미하는 것으로, 두 가지 클래스에만 공개한다.

같은 패키지의 모든 클래스

다른 패키지라도 자식 클래스

자식 클래스는 9주차 상속에서 배운다.

같은 패키지의 C 와 다른 패키지의 자식 D 가 쓸 수 있다

B 와 같은 패키지

class B

protected
멤버의 주인

class C

같은 패키지
클래스



다른 패키지

class D

다른 패키지
자식 클래스



class A

다른 패키지
클래스



접근 지정자를 안 쓴 멤버는 같은 패키지 안에서만 쓴다

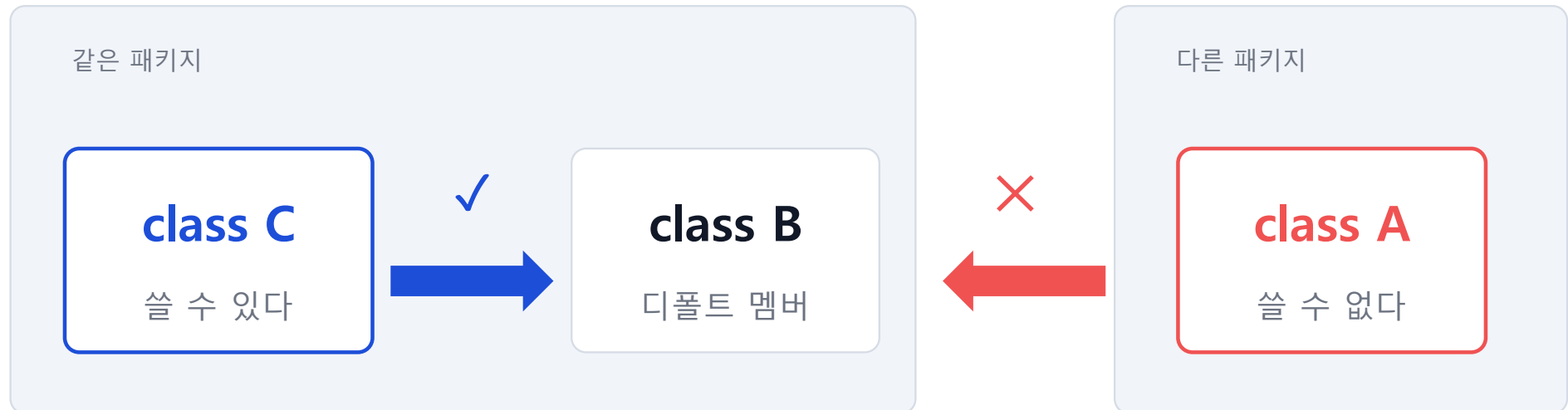
접근 지정자가 생략된 멤버의 경우, 디폴트(default) 멤버라고 한다.

동일한 패키지 내에 있는 클래스들만 디폴트 멤버를 자유롭게 접근할 수 있다.

다른 이름

`package-private`

같은 패키지의 C 는 쓰고 다른 패키지의 A 는 못 쓴다



문제: 네 필드 가운데 어느 줄이 오류일까

다음 코드의 두 클래스 SampleClass와 AccessEx 클래스는 AccessEx.java에 들어 있어 컴파일 되면, 동일한 패키지에 저장된다. 컴파일 오류를 찾아내고 이유를 설명하라.

예제 4-10 접근 지정자가 다른 필드 넷

Ex06_AccessEx.java

```
class SampleClass {  
    public int field1;  
    protected int field2;  
    int field3;  
    private int field4;  
}
```

예제 4-10 네 필드에 값 넣기

Ex06_AccessEx.java

```
public class Ex06_AccessEx {  
    public static void main(String[] args) {  
        SampleClass s = new SampleClass();  
        s.field1 = 0;  
        s.field2 = 1;  
        s.field3 = 2;  
        s.field4 = 3;  
    }  
}
```

컴파일 오류 원문

Ex06_AccessEx.java:14: error: field4 has private access in SampleClass

s.field4 = 3;

^

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

14번 줄 field4 에서 막혔다

field4 만 private 이라 SampleClass 밖에서 막힌다

field1

public

된다

field2

protected

된다(같은 패키지)

field3

디폴트

된다(같은 패키지)

field4

private

안 된다

field4 줄을 빼고 남은 셋 쓰기

Ex07_AccessFix.java

```
class AccessSample {  
    public int field1;  
    protected int field2;  
    int field3;  
    private int field4;  
}
```

세 필드 값 출력하기

Ex07_AccessFix.java

```
public class Ex07_AccessFix {  
    public static void main(String[] args) {  
        AccessSample s = new AccessSample();  
        s.field1 = 0;  
        s.field2 = 1;  
        s.field3 = 2;  
        System.out.println(s.field1 + " " + s.field2 + " " + s.field3);  
    }  
}
```

0 1 2

0 1 2

public · protected · 디폴트는 같은 패키지에서 된다

같은 파일 안에서 public · protected · 디폴트 필드 읽기

Ex08_SameFileAccess.java

```
class Box {  
    public int a = 1;  
    protected int b = 2;  
    int c = 3;  
}  
  
public class Ex08_SameFileAccess {  
    public static void main(String[] args) {  
        Box x = new Box();  
        System.out.println(x.a + " " + x.b + " " + x.c);  
    }  
}
```

1 2 3

1 2 3

같은 패키지 줄은 0 0 0

실행 결과를 표 4-1 첫 행과 맞춰 보기

	private	디폴트	protected	public
같은 패키지의 클래스	X	O	O	O
오늘 실행	Ex06 field4 → 오류		Ex08 a b c → 1 2 3	
다른 패키지의 클래스	13주차			

필드를 private 으로 두면 값은 메소드를 거쳐서만 바뀐다

필드가 디폴트

```
s.value = 150;
```

아무 값이나 들어간다

필드가 private

```
s.setValue(150);
```

메소드가 값을 보고 거른다

값을 읽는 getter, 바꾸는 setter

getter

get필드이름() — 값을 돌려준다

```
public int getRadius() { return this.radius; }
```

setter

set필드이름(값) — 값을 바꾼다

```
public void setRadius(int radius) { this.radius = radius; }
```

필드는 private, 메소드는 public

필드는 감추고 그 필드를 다루는 메소드는 연다.

필드

```
private int value;
```

메소드

```
public void setValue(int v)
```

setter 로 점수 범위를 거르는 Score

Ex09_ScoreGuard.java

```
class Score {  
    private int value;  
  
    public void setValue(int v) {  
        if (v < 0 || v > 100) {  
            System.out.println("잘못된 점수: " + v);  
            return;  
        }  
        value = v;  
    }  
  
    public int getValue() {  
        return value;  
    }  
}
```

85 와 150 을 차례로 넣어 보기

Ex09_ScoreGuard.java

```
public class Ex09_ScoreGuard {  
    public static void main(String[] args) {  
        Score s = new Score();  
        s.setValue(85);  
        s.setValue(150);  
        System.out.println(s.getValue());  
    }  
}
```

150 은 거르고 85 가 남는다

잘못된 점수: 150
85

잘못된 값은 setter 에서 멈췄다

검사하는 자리가 setter 하나로 모인다



점수 범위가 바뀌어도 setValue 한 곳만 고친다.

이름과 나이를 감춘 Pet

Ex10_PetGetter.java

할 일: `getAge()` 가 나이를 돌려주게 고친다.

```
class Pet {  
    private String name;  
    private int age;  
  
    public Pet(String n, int a) {  
        name = n;  
        age = a;  
    }  
}
```

getName 을 보고 getAge 를 채워 보기

Ex10_PetGetter.java

getName 과 같은 꼴로 getAge 를 채운다.

```
public String getName() {  
    return name;  
}  
  
public int getAge() {  
    // 여기서 부터 코드를 작성하세요.  
    return 0; // 이 줄을 고치세요.  
    // 여기 까지  
}  
}
```

반려동물 하나를 만들어 출력하기

Ex10_PetGetter.java

```
public class Ex10_PetGetter {  
    public static void main(String[] args) {  
        Pet p = new Pet("초코", 3);  
        System.out.println(p.getName() + " " + p.getAge() + "살");  
    }  
}
```

초코 3살

초코 3살

private 필드를 getter 로 읽었다

main 에서 p.age 라고 쓰면 컴파일 오류다

```
System.out.println(p.age);
```

age has private access in Pet

감춘 필드는 getter 로만 읽는다.

private 메소드를 밖에서 부르면

Ex11_PrivateMethod.java

```
class Safe {  
    private void open() {  
        System.out.println("금고가 열렸다");  
    }  
}  
  
public class Ex11_PrivateMethod {  
    public static void main(String[] args) {  
        Safe s = new Safe();  
        s.open();  
    }  
}
```

컴파일 오류 원문

Ex11_PrivateMethod.java:10: error: open() has private access in Safe

s.open();

^

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

메소드도 private 이면 밖에서 못 부른다

private 메소드는 클래스 안의 도우미로 쓴다

밖에 보일 필요 없는 일을 맡기는 메소드는 private 으로 둔다.

바깥에서 부른다

```
public void open()
```



클래스 안에서만 부른다

```
private boolean check()
```

온도 조절기의 setter 에 범위 검사 넣어 보기

Ex12_TempSetter.java

10 보다 작거나 30 보다 큰 값이면 알리고 바꾸지 않는다.

```
class Thermostat {  
    private int temp = 20;  
  
    public void setTemp(int t) {  
        // 여기서 부터 코드를 작성하세요.  
        temp = t;  
        // 여기 까지  
    }  
  
    public int getTemp() {  
        return temp;  
    }  
}
```

25 와 99 를 차례로 넣어 보기

Ex12_TempSetter.java

```
public class Ex12_TempSetter {  
    public static void main(String[] args) {  
        Thermostat th = new Thermostat();  
        th.setTemp(25);  
        th.setTemp(99);  
        System.out.println("지금 온도 " + th.getTemp());  
    }  
}
```

99 는 거르고 25 가 남는다

설정할 수 없는 온도: 99
지금 온도 25

setter 가 범위 밖 값을 막았다

빈칸 그대로 실행하면

지금 온도 99

검사가 없으면 99 도 들어간다

2교시에서 본 것

1

멤버의 공개 범위는
private < 디폴트 < protected < public

2

private 멤버는 그 클래스 안에서
만 쓴다

3

같은 패키지에서는 private 만 막힌다

4

필드는 private,
다루는 메소드는 public

2교시 실습 살펴보기

실습 ②

getter 를 두면 main 은
p.age 가 아니라
p.getAge() 로 읽는다

p.getAge()

실습 ③

setter 에서 return 으로
돌아가면 아래 temp = t 가
실행되지 않는다

return

3교시

static 과 final

모두가 함께 쓰는 멤버와 바꿀 수 없는 상수를 만든다.

static 멤버

static 메소드

final 상수

지난주 예제의 static 을 다시 보기

```
static void increase(int m) { ... }
```

지난주에는 main 에서 바로 부르려고 붙인 말이라고만 했다.

static 이 붙으면 객체 없이도 쓸 수 있다

static 없음

객체를 만든 뒤
객체.메소드()

static 있음

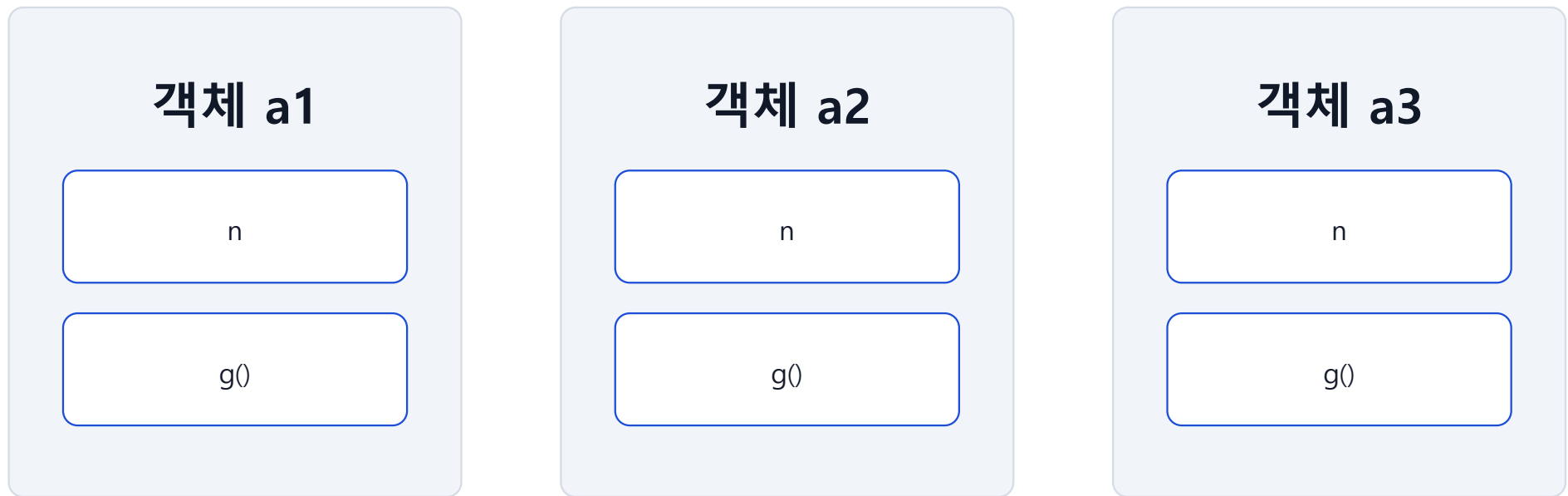
객체 없이
클래스 이름.메소드()

main 도 static 이다. 그래서 객체 없이 시작된다.

static 이 붙은 필드와 메소드가 static 멤버다

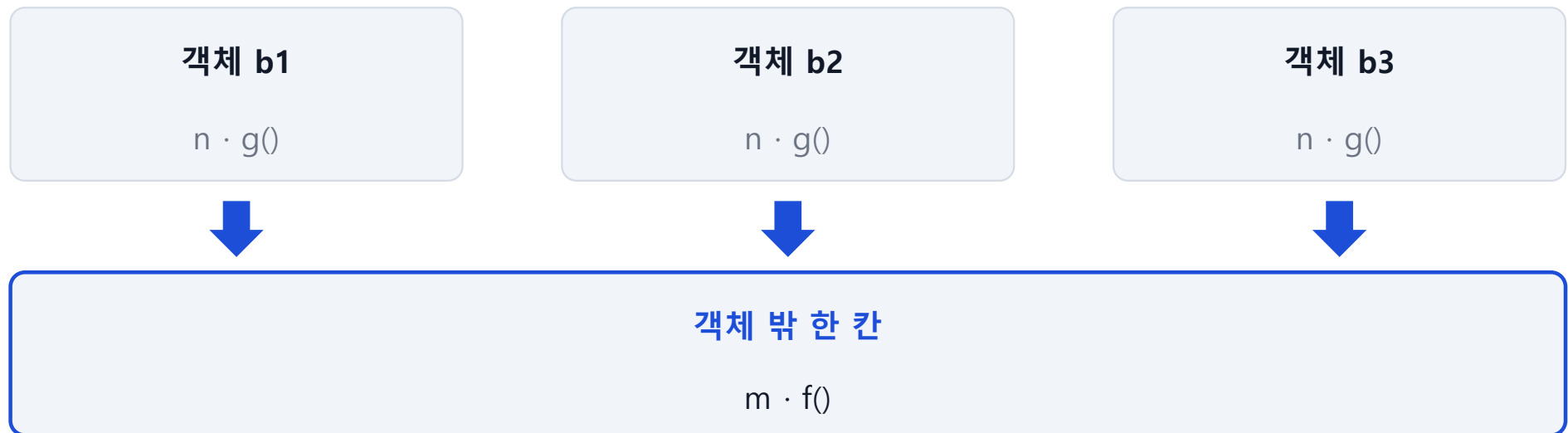
```
class StaticSample {  
    int n; // non-static 필드  
    void g() {...} // non-static 메소드  
    static int m; // static 필드  
    static void f() {...} // static 메소드  
}
```

non-static 멤버는 객체마다 따로 생긴다



객체가 늘면 멤버도 늘어난다

static 멤버는 객체 밖에 하나만 있고
모두가 함께 쓴다



클래스 멤버

클래스당 하나만 생겨 모든 객체가 공유하는 static 멤버

static 멤버는 클래스당 하나만 생성되는 멤버로서, 동일한 클래스의 모든 객체들이 공유하므로 클래스 멤버라고 부른다.

인스턴스 멤버

객체마다 따로 생기는 non-static 멤버

반면, non-static 멤버는 각 객체마다
별도로 생기므로 인스턴스 멤버라고
부른다.

객체 . 멤버

객체 . 멤버

객체 . 멤버

빵을 만들 때마다 static count 를 올리는 Bread

Ex13_StaticCount.java

```
class Bread {  
    int no;  
    static int count;  
  
    Bread() {  
        count++;  
        no = count;  
    }  
}
```

빵 세 개를 만들고 번호와 count 출력하기

Ex13_StaticCount.java

```
public class Ex13_StaticCount {  
    public static void main(String[] args) {  
        Bread b1 = new Bread();  
        Bread b2 = new Bread();  
        Bread b3 = new Bread();  
        System.out.println(b1.no + " " + b2.no + " " + b3.no);  
        System.out.println(Bread.count);  
    }  
}
```

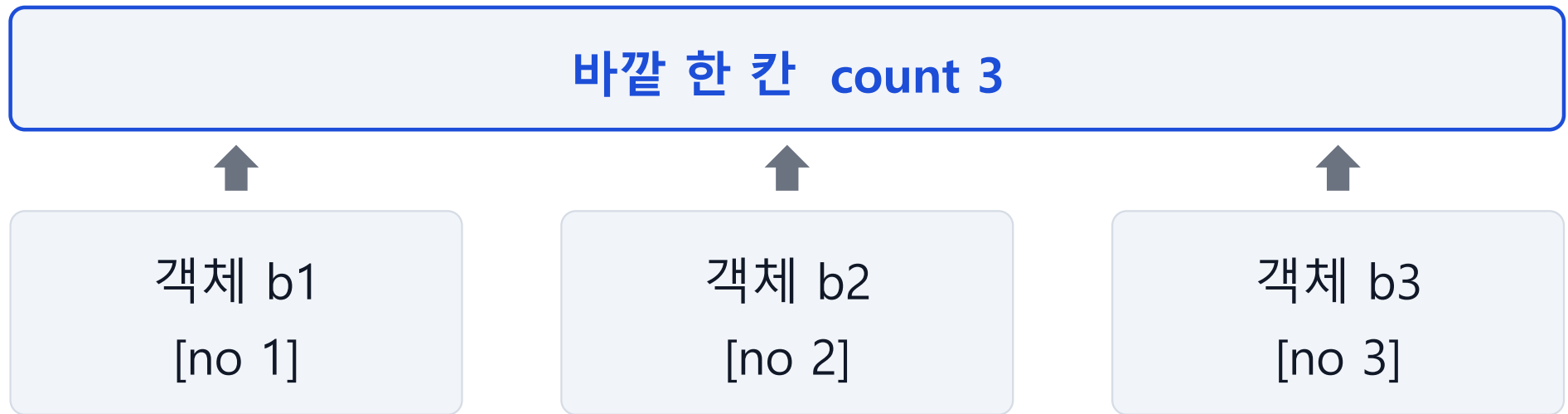


1 2 3, 그리고 3

1 2 3
3

no 는 셋, count 는 하나

count 는 하나, no 는 셋



세 생성자가 같은 count 를 하나씩 올렸다.

표 4-2 non-static 멤버와 static 멤버의 차이

구분	non-static 멤버	static 멤버
공간적 특성	멤버는 객체마다 별도 존재 · 인스턴스 멤버라고 부름	멤버는 클래스당 하나 생성 · 멤버는 객체 내부가 아닌 별도의 공간에 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

static 멤버는 객체가 생기기 전에도 있다

static 멤버는 프로그램을 시작할 때나 클래스 로딩 시에 생성된다.

그러므로 객체를 생성하기 전에도 static 멤버는 사용할 수 있다.

static 멤버는 프로그램이 끝날 때 사라진다

non-static

객체가 사라지면
함께 사라진다

static

객체가 사라져도 남고
프로그램이 끝날 때 사라진다

클래스 이름으로 쓴다

```
StaticSample.m = 3; // 클래스 이름으로 static 필드 접근  
StaticSample.f(); // 클래스 이름으로 static 메소드 호출
```

객체를 만들지 않고 클래스 이름으로 쓴다.

객체 이름으로도 쓸 수 있다

```
StaticSample b1 = new StaticSample();  
b1.m = 3; // 객체 이름으로 static 필드 접근  
b1.f(); // 객체 이름으로 static 메소드 호출
```

b1 으로 바꾼 m 을 b2 로 읽어 보기

Ex14_StaticSample.java

```
class StaticSample {  
    int n;  
    static int m;  
}  
  
public class Ex14_StaticSample {  
    public static void main(String[] args) {  
        StaticSample.m = 3; // 클래스 이름으로 static 필드 접근  
        StaticSample b1 = new StaticSample();  
        StaticSample b2 = new StaticSample();  
        b1.m = 5; // 객체 이름으로 static 필드 접근  
        System.out.println(b2.m);  
        System.out.println(StaticSample.m);  
    }  
}
```

5 와 5

5
5

b1 으로 바꿨는데 b2 도 5

static 멤버는 클래스 이름으로 쓰는 편이 읽기 쉽다

StaticSample.m 이라고 쓰면 모두가 함께 쓰는 칸이라는 것이 바로 보인다.

권장: 클래스 이름.멤버

non-static 필드를 클래스 이름으로 쓰면

Ex15_StaticError.java

```
class Sample15 {  
    int n;  
    static int m;  
}  
  
public class Ex15_StaticError {  
    public static void main(String[] args) {  
        Sample15.m = 3;  
        Sample15.n = 5;  
    }  
}
```

n 은 non-static

컴파일 오류 원문

Ex15_StaticError.java:9: error: non-static variable n cannot be referenced from a static context

Sample15.n = 5;

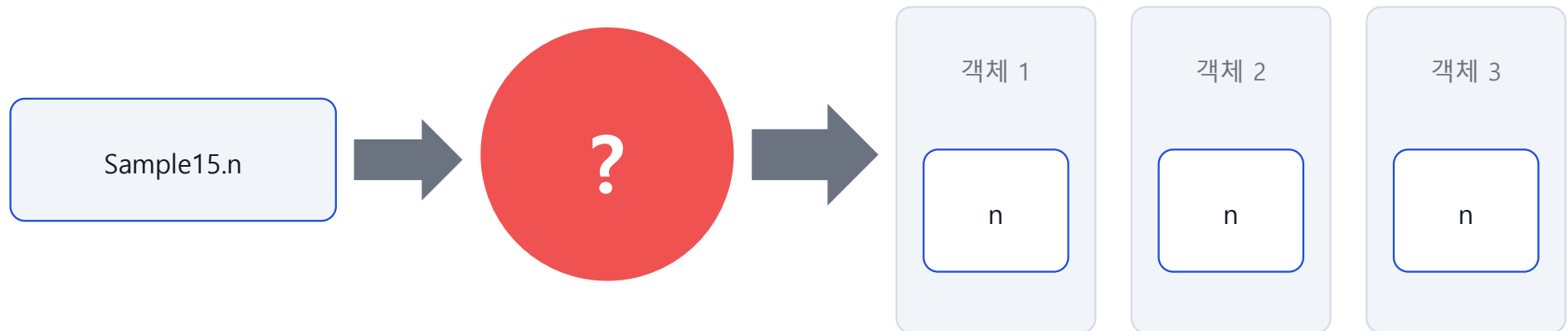
^

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

n 은 객체가 있어야 생기는 필드다

어느 객체의 n 인지 정할 수 없다



n 은 객체마다 따로라 클래스 이름만으로는 어느 n 인지 모른다.

static next 로 자리 번호를 매겨 보기

Ex16_SeatNumber.java

자리를 하나 만들 때마다 next 를 하나 늘린다.

```
class Seat {  
    static int next = 1;  
    int no;  
  
    Seat() {  
        no = next;  
        // 여기서 부터 코드를 작성하세요.  
  
        // 여기 까지  
    }  
}
```

자리 셋을 만들고 번호 출력하기

Ex16_SeatNumber.java

```
public class Ex16_SeatNumber {  
    public static void main(String[] args) {  
        Seat s1 = new Seat();  
        Seat s2 = new Seat();  
        Seat s3 = new Seat();  
        System.out.println(s1.no + " " + s2.no + " " + s3.no);  
        System.out.println("다음 번호 " + Seat.next);  
    }  
}
```

1 2 3, 다음 번호 4

1 2 3
다음 번호 4

세 자리가 같은 next 를 나눠 썼다

빈칸 그대로 실행하면

1 1 1
다음 번호 1

늘리지 않으면 모두 1 번

next 에서 static 을 지우면 모두 1 번이 된다

```
int next = 1;
```

객체마다 따로

객체마다 next 가 1 에서 시작하니 번호가 늘지 않는다. 그리고 Seat.next 줄은 컴파일 오류가 된다.

static 메소드는 어디서나 부르는 함수처럼 쓴다

응용프로그램 작성 시 모든 클래스에서 공유하는 전역 변수(global variable)나 모든 클래스에서 호출할 수 있는 전역 함수(global function)를 만들고자 할 때가 있다.

이런 경우 static으로 선언하면 된다.

객체 없이 부르는 계산 도우미 Calc

Ex17_CalcStatic.java

```
class Calc {  
    static int abs(int a) {  
        return a > 0 ? a : -a;  
    }  
  
    static int max(int a, int b) {  
        return a > b ? a : b;  
    }  
}
```

Calc.abs 와 Calc.max 부르기

Ex17_CalcStatic.java

```
public class Ex17_CalcStatic {  
    public static void main(String[] args) {  
        System.out.println(Calc.abs(-5));  
        System.out.println(Calc.max(10, 8));  
        System.out.println(Math.max(10, 8));  
    }  
}
```

5, 10, 10

```
5  
10  
10
```

우리가 만든 max 와 자바의 Math.max 가 같은 답

Math 의 메소드는 모두 static 이다

`Math.max(a, b)`

`Math.abs(a)`

`Math.sqrt(x)`

`Math.random()`

new Math() 를 하지 않고 바로 부른다.

분과 초를 초로 바꾸는 static 메소드 채워 보기

Ex18_TimeUtil.java

toSeconds 가 분 × 60 + 초 를 돌려주게 고친다.

```
class TimeUtil {  
    static int toSeconds(int min, int sec) {  
        // 여기서 부터 코드를 작성하세요.  
        return 0; // 이 줄을 고치세요.  
        // 여기 까지  
    }  
}  
  
public class Ex18_TimeUtil {  
    public static void main(String[] args) {  
        System.out.println(TimeUtil.toSeconds(2, 5) + "초");  
        System.out.println(TimeUtil.toSeconds(10, 0) + "초");  
    }  
}
```



125초, 600초

125초
600초

객체 없이 계산했다

빈칸 그대로 실행하면

0초

0초

자리표시 | return 0 이 그대로

static 메소드는 static 멤버만 쓸 수 있다

static 메소드는 오직 static 멤버만 접근할 수 있다.

인스턴스 메소드는 static 멤버를 쓸 수 있다. 거꾸로는 안 된다.

static 메소드 s1 이 non-static 필드 n 을 쓰면

Ex19_StaticMethod.java

```
class StaticMethod {  
    int n;  
    static int m;  
  
    static void s1(int x) {  
        n = x; // static 메소드에서 non-static 필드 사용  
    }  
}  
  
public class Ex19_StaticMethod {  
    public static void main(String[] args) {  
        StaticMethod.s1(3);  
    }  
}
```

컴파일 오류 원문

Ex19_StaticMethod.java:6: error: non-static variable n cannot be referenced from a static context

```
n = x; // static 메소드에서 non-static 필드 사용
```

^

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

Sample15 와 같은 문장이다

static 메소드에서는 this 를 쓸 수 없다

static 메소드에서는 this를 사용할 수 없다.

```
static void f() { this.n = x; }  
// 오류. static 메소드에서는 this 사용 불가능
```

static 메소드는 객체 없이 불러서 this 가 없다.

문제: 잘못 쓴 static 고르기

11. 다음 멤버 함수의 사용이 잘못된 것은?

```
class StaticTest {  
    static int a;  
    static int getA() { return a; } // ①  
    int b;  
    int getB() { return b; } // ②  
    int f() { return getA(); } // ③  
    static int g() { return getB(); } // ④  
}
```

12. 다음 코드에서 잘못된 문장은?

```
StaticSample.x = 5; // ①  
StaticSample.y = 10; // ②  
int tmp = StaticSample.f(); // ③  
StaticSample a = new StaticSample();  
tmp = a.y; // ④
```

풀이: 11번은 ④, 12번은 ①

11번 정답 ④

static int g() 가 non-static 메소드 getB()
를 부른다

12번 정답 ①

non-static 필드 x 를 클래스 이름으로 썼
다

두 문제 모두 오늘 본 오류 예제와 같은 풀이다.

final 은 클래스, 메소드, 필드에 붙는다

final 키워드는 3가지 용도로 사용된다.



final 클래스

상속 불가



final 메소드

오버라이딩 불가



final 필드

값 변경 불가

final 클래스는 물려줄 수 없다

```
final class FinalClass {  
}  
class SubClass extends FinalClass { // 컴파일 오류 발생  
}
```

상속(extends)은 9주차에 배운다.

final 메소드는 자식이 고쳐 쓸 수 없다

```
protected final int finalMethod() { ... }
```

자식이 메소드를 고쳐 쓰는 오버라이딩은 11주차에 배운다.

final 필드는 상수다. 한 번 정하면 못 바꾼다

상수가 된다

자바에서 final로 필드를 선언하면 필드는 상수가 된다.



값을 못 바꾼다

상수 필드는 한 번 초기화되면 값을 변경할 수 없다.

final 필드 ROWS 에 다시 값을 넣으면

Ex20_FinalField.java

```
class FinalFieldClass {  
    final int ROWS = 10; // 상수 선언. 초깃값(10) 지정  
  
    void f() {  
        int[] intArray = new int[ROWS]; // 상수 활용  
        ROWS = 30; // final 필드 값은 바꿀 수 없다  
    }  
}  
  
public class Ex20_FinalField {  
    public static void main(String[] args) {  
        new FinalFieldClass().f();  
    }  
}
```

컴파일 오류 원문

Ex20_FinalField.java:6: error: cannot assign a value to final variable ROWS

ROWS = 30; // final 필드 값은 바꿀 수 없다

^

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

상수에는 다시 넣을 수 없다

public static final 로 모두가 쓰는 상수 PI

Ex21_SharedPI.java

```
class SharedClass {  
    public static final double PI = 3.14;  
}  
  
public class Ex21_SharedPI {  
    public static void main(String[] args) {  
        double radius = 10;  
        double area = SharedClass.PI * radius * radius;  
        System.out.println(area);  
        System.out.println(Math.PI);  
    }  
}
```

314.0 과 3.141592653589793

314.0

3.141592653589793

우리 상수는 3.14, 자바의 Math.PI 는 훨씬 길다



☕ 원주율을 법으로 정할 뻔한 1897년

1897년 미국 인디애나주 하원은 원의 넓이를 새로 구하는 법을 「새로운 수학적 진리」로 정하자는 법안 246호를 반대 없이 통과시켰다.

퍼듀 대학 수학 교수가 상원 의원들을 설득했고, 상원은 1897년 2월 12일 법안을 무기한 미뤘다.

법안 문장("the ratio of the diameter and circumference is as five-fourths to four")대로 계산하면 원주율은 3.2 가 된다.

그림: Rock Island Argus 시사만화(1897-03-06) · 퍼블릭 도메인

상수는 바꿀 수 없으니 처음 넣는 값이 맞아야 한다

```
public static final double PI = 3.14;
```

한 번 정하면 끝

더 정확한 값이 필요하면 직접 쓰지 말고 Math.PI 를 쓴다.

3교시에서 본 것

1

static 멤버는 클래스당 하나, 모든 객체가 함께 쓴다

2

static 멤버는 객체 없이 클래스 이름으로 쓴다

3

non-static 멤버는 클래스 이름으로 쓸 수 없다

4

static 메소드는 static 멤버만 쓰고 this 를 쓸 수 없다

5

final 필드는 값을 바꿀 수 없는 상수다

3교시 실습 살펴보기



next++ 를 빠뜨리면 자리 셋이 모두 1 번이다.
static 은 함께 쓰는 칸 하나다(실습 ④)



TimeUtil.toSeconds 는 객체 없이 불렀다.
계산만 하는 메소드는 static 으로 둔다(실습 ⑤)

안 될 때 여기부터

틀림

- private 필드를 main 에서 바로 씀
- static 메소드에서 인스턴스 필드
- final 필드에 다시 값을 넣음

맞음

- ✓ getter · setter 를 거친다
- ✓ 객체를 만들어 쓴다
- ✓ final 을 뺀다

오류 줄 끝의 **private · static · final** 낱말을 먼저 읽는다

오늘 공부한 내용

참조하는 레퍼런스가 하나도 없는 객체가 가비지다

가비지는 JVM 이 가비지 컬렉션으로 거두고, 언제 거둘지는 알 수 없다

공개 범위는 private < 디폴트 < protected < public

static 멤버는 클래스당 하나, 모든 객체가 함께 쓴다

final 필드는 한 번 정하면 바꿀 수 없는 상수다

다음 주엔 중간고사

1주차부터 7주차까지 배운 것을 객관식으로 봅니다



다음 주 예고

- 중간고사
- 1~7주차 · 객관식