



경기과학기술대학교 · 인공지능학과

자바 프로그래밍

6주차

생성자와 메소드 오버로딩

만드는 순간 준비를 마치고,
같은 이름으로 여러 일을 한다.

오늘의 강의

1 교시 생성자 만들기, 2 교시 기본 생성자와 this, 3 교시 인자 전달과 오버로딩

1교시



생성자 만들기

new 와 함께 불린다
리턴 타입이 없다

2교시



기본 생성자와 this

생성자가 없을 때
this 는 객체 자신

3교시



인자 전달과 오버로딩

값과 레퍼런스
같은 이름, 다른 매개 변수

1교시

생성자 만들기

객체를 만드는 순간 필드에 값을 넣는다.

생성자 선언

특징 다섯

5주차 코드 줄이기

지난주에 세 줄로 만든 객체

```
book[i] = new Book(); // 배열 원소 객체 생성  
book[i].title = title; // 제목 필드에 값 넣기  
book[i].author = author; // 저자 필드에 값 넣기
```

```
sq[i] = new Square(); // i 번째 원소 객체 생성  
sq[i].side = i+1; // 한 변의 길이를 i+1 로 설정
```

오늘은 이 줄을 한 줄로 줄인다.

객체를 먼저 만들고 값은 나중에 넣었다

1단계 만들기

```
new Book()
```

필드는 기본값(null · 0)



2단계 채우기

```
book[i].title = ...
```

필드에 값이 들어간다

채우는 줄을 빠뜨리면 값은 null 이나 0 으로 남는다.

생성자

객체가 생성될 때 초기화를 위해 실행되는 메소드

정의

생성자는 객체가 생성될 때 초기화를 위해 실행되는 메소드이다.

초기화는 객체가 쓰이기 전에 필드에 첫 값을 넣는 일이다.

맨 얼굴 객체와 화장한 객체



생성자를 호출하여 화장하는 얼굴 부분이나 색을 다르게 하면, 서로 다른 모양의 얼굴 객체를 생성할 수 있다.



생성자는 거의 모든 객체 지향 언어에 있다

생성자 개념은 자바, C++, C# 등 거의 모든 객체 지향 언어에 존재하며 사용하는 방법 또한 거의 비슷하다.

자바

C++

C#



생성자의 옛 이름은 new 함수였다

C with Classes · 1979년 시작

C++의 전신인 C with Classes는 1980년 첫 구현에서 생성자를 `void new();` 로 적었다.

```
void new();
```

당시 이름은 「new 함수」였는데 헛갈린다는 말이 많아 C++로 오면서 constructor(생성자)라는 이름을 붙였다.



new 다음의 괄호가 생성자를 부른다

```
Circle pizza = new Circle(10, "자바피자");
```



10 → r

"자바피자" → n

```
public Circle(int r, String n) { ... }
```

new 는 객체를 만든다. 괄호 안 값은 생성자에게 간다.

예제 4-3 두 개의 생성자를 가진 Circle 클래스

```
class Circle {  
    int radius;  
    String name;  
  
    public Circle() { // 매개 변수 없는 생성자  
        radius = 1; name = ""; // radius의 초기값은 1  
    }  
  
    public Circle(int r, String n) { // 매개 변수를 가진 생성자  
        radius = r; name = n;  
    }  
    public double getArea() {  
        return 3.14*radius*radius;  
    }  
}
```

Ex01_CircleCtor.java

생성자 이름은 클래스 이름과 동일

생성자는 리턴 타입 없음

생성자는 radius와 name 필드 초기화

예제 4-3 두 생성자로 객체 만들기

Ex01_CircleCtor.java

```
public class Ex01_CircleCtor {  
    public static void main(String[] args) {  
        Circle pizza = new Circle(10, "자바피자"); // Circle 객체 생성, 반지름 10  
        double area = pizza.getArea();  
        System.out.println(pizza.name + "의 면적은 " + area);  
  
        Circle donut = new Circle(); // Circle 객체 생성, 반지름 1  
        donut.name = "도넛피자";  
        area = donut.getArea();  
        System.out.println(donut.name + "의 면적은 " + area);  
    }  
}
```

예제 4-3 실행 결과

자바피자의 면적은 314.0

도넛피자의 면적은 3.14

넘긴 값이 다르니 넓이도 다르다

어느 new 가 어느 생성자를 불렀나

main 의 new

`new Circle(10, "자바피자")`

값 둘

`new Circle()`

값 없음



불리는 생성자

`public Circle(int r, String n)`

매개 변수 둘

`public Circle()`

매개 변수 없음

괄호 안 값의 개수와 타입을 보고 맞는 생성자가 불린다.

생성자 이름은 클래스 이름과 같다

생성자의 이름은 클래스 이름과 동일하다.

그래서 다른 메소드와 쉽게 구분된다.

클래스 선언

```
class Circle
```

=

생성자 선언

```
public Circle()
```

생성자는 여러 개 둘 수 있다

- 생성자를 여러 개 작성할 수 있다.
- 매개 변수의 개수와 타입만 다르다면, 클래스 내에 같은 이름의 생성자를 여러 개 둘 수 있다.

```
public class Circle {  
    public Circle() {...} // 매개 변수 없는 생성자  
    public Circle(int r, String n) {...} // 2개의 매개 변수를 가진 생성자  
}
```

생성자는 객체를 만들 때 한 번만 불린다

생성자는 객체를 생성할 때 한 번만 호출된다.

```
Circle pizza = new Circle(10, "자바피자"); // 생성자 Circle(int r, String n) 호출  
Circle donut = new Circle(); // 생성자 Circle() 호출
```

`new` 가 실행되는 순간 한 번만 불린다.

생성자에는 리턴 타입이 없다

- 생성자에 리턴 타입을 지정할 수 없다.
- 리턴할 값이 없다고 해서, void를 리턴 타입으로 지정해서도 안 된다.

```
public Circle() {...}  
public void Circle() {...} // 오류. void를 리턴 타입으로 사용할 수 없음
```

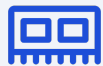
생성자는 객체가 일하기 전의 준비를 맡는다

생성자의 목적은 객체가 생성될 때, 필요한 초기 작업을 위함이다.

준비 네 가지



필드에 초깃값
넣기



메모리 확보



파일 열기



네트워크 연결

생성자 특징 다섯 줄 정리

특징

한 줄 예

1 생성자의 이름은 클래스 이름과 동일하다.

class Circle → public Circle()

2 생성자를 여러 개 작성할 수 있다.

Circle() · Circle(int r, String n)

3 생성자는 객체를 생성할 때 한 번만 호출된다.

new Circle() 할 때

4 생성자에 리턴 타입을 지정할 수 없다.

void 도 쓰지 않는다

5 생성자의 목적은 객체가 생성될 때, 필요한 초기 작업을 위함이다.

필드에 첫 값 넣기

예제 4-4 생성자 둘을 가진 Book 클래스

```
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;  
    }  
}
```

Ex02_BookCtor.java

제목만 받는 생성자는
저자를 작자미상으로
넣는다.

예제 4-4 생성자를 골라 책 두 권 만들기

Ex02_BookCtor.java

```
public class Ex02_BookCtor {  
    public static void main(String[] args) {  
        Book javaBook = new Book("Java", "황기태"); // 생성자 Book(String t, String a) 호출  
        Book bible = new Book("Bible"); // 생성자 Book(String t) 호출  
        System.out.println(javaBook.title + " " + javaBook.author);  
        System.out.println(bible.title + " " + bible.author);  
    }  
}
```

예제 4-4 실행 결과

Java 황기태
Bible 작자미상

값을 하나 넘기면 저자는 생성자가 정한다

세 줄이 한 줄이 됐다

5주차

```
book[i] = new Book();  
book[i].title = title;  
book[i].author = author;
```

오늘

```
Book javaBook = new Book("Java", "황기태");
```

만들기와 채우기를 생성자가 한 번에 한다.

생성자 하나로 휴대폰 두 대 만들기

Ex03_PhoneCtor.java

```
class Phone {  
    String model;  
    int price;  
  
    public Phone(String m, int p) {  
        model = m;  
        price = p;  
    }  
}
```

두 대의 모델과 가격 출력하기

Ex03_PhoneCtor.java

```
public class Ex03_PhoneCtor {  
    public static void main(String[] args) {  
        Phone a = new Phone("갤럭시", 1200000);  
        Phone b = new Phone("아이폰", 1500000);  
        System.out.println(a.model + " " + a.price + "원");  
        System.out.println(b.model + " " + b.price + "원");  
    }  
}
```

new Phone() 괄호 안 값은 모델, 가격 순서로 쓴다.

휴대폰 두 대 결과

갤럭시 1200000원
아이폰 1500000원

생성자는 하나, 객체는 둘

연필 생성자 몸통 채워 보기

Ex04_PencilCtor.java

생성자 안에서 매개 변수 len 을 필드 length 에 넣는다.

```
class Pencil {  
    String color;  
    int length;  
  
    public Pencil(String c, int len) {  
        color = c;  
        // 여기서 부터 코드를 작성하세요.  
  
        // 여기 까지  
    }  
}
```

연필 하나 만들어 출력하기

Ex04_PencilCtor.java

```
public class Ex04_PencilCtor {  
    public static void main(String[] args) {  
        Pencil p = new Pencil("빨강", 17);  
        System.out.println(p.color + " 연필, 길이 " + p.length + "cm");  
    }  
}
```

new Pencil("빨강", 17) 의 17 이 매개 변수 len 으로 간다.

연필 결과

빨강 연필, 길이 17cm

생성자가 색과 길이를 모두 넣었다

지난주 정사각형 배열은 두 줄로 만들었다

```
sq[i] = new Square(); // i 번째 원소 객체 생성  
sq[i].side = i+1; // 한 변의 길이를 i+1 로 설정
```

오늘 Square 에는 한 변을 받는 생성자가 있다.

생성자가 있는 Square 클래스

Ex05_SquareCtor.java

```
class Square {  
    int side;  
  
    public Square(int s) {  
        side = s;  
    }  
  
    public int getArea() {  
        return side*side;  
    }  
}
```

정사각형 다섯 개를 생성자로 만들어 보기

반복 안에서 한 번이 $i+1$ 인 Square 객체를 생성자로 만들어 `sq[i]` 에 넣는다.

```
public class Ex05_SquareCtor {  
    public static void main(String[] args) {  
        Square[] sq = new Square[5];  
        for(int i=0; i<sq.length; i++) {  
            // 여기서 부터 코드를 작성하세요.  
  
            // 여기 까지  
        }  
        for(int i=0; i<sq.length; i++)  
            System.out.print(sq[i].getArea() + " ");  
    }  
}
```

정사각형 넓이 결과

1 4 9 16 25

지난주와 같은 결과, 줄은 하나 줄었다

빈칸 그대로 실행하면

Exception in thread "main" java.lang.NullPointerException: Cannot invoke "Square.getArea()" because "<local1>[<local2>]" is null

at Ex05_SquareCtor.main(Ex05_SquareCtor.java:22)

객체를 만들지 않은 칸을 쓰려 했다

void 를 붙인 생성자는 메소드가 된다

Ex06_VoidCtor.java

```
class Pen {  
    String color;  
  
    public void Pen() { // 생성자처럼 보이지만 메소드다  
        color = "파랑";  
    }  
}
```

```
public class Ex06_VoidCtor {  
    public static void main(String[] args) {  
        Pen p = new Pen();  
        System.out.println("펜 색: " + p.color);  
    }  
}
```

펜 색이 null 로 나온다

펜 색: null

파란을 넣는 줄은 실행되지 않았다

new Pen() 은 무엇을 불렀나

`public void Pen()`



리턴 타입이 있으니 생성자가 아닌 메소드

`new Pen()`



컴파일러가 넣어 둔 빈 생성자를 불렀다

생성자에는 리턴 타입을 쓰지 않는다.

1교시에서 본 것

1

생성자는 객체가 생성될 때 초기화를 위해 실행된다

2

생성자 이름은 클래스 이름과 같고 리턴 타입이 없다

3

new 다음 괄호 안 값이 생성자의 매개 변수로 간다

4

생성자는 여러 개 둘 수 있다

1교시 실습 살펴보기



기본값 필드

생성자 안에서 놓지 않은 필드는 기본값으로 남는다(연필 길이 0cm)



원소마다 new

배열을 만든 뒤 원소마다 new 로 객체를 만들어야 한다(빈칸 그대로면 NullPointerException)

2교시

기본 생성자와 this

생성자가 없을 때와 자기 객체를 가리킬 때를 본다.

기본 생성자

this

this()

기본 생성자 - 매개 변수도 실행 코드도 없는 생성자

기본 생성자란 매개 변수가 없고 또한 실행 코드가 없어 아무 일도 하지 않고 단순 리턴하는 생성자이다.

디폴트 생성자라고도 부른다.

기본 생성자는 이렇게 생겼다

```
class Circle {  
    public Circle() { } // 기본 생성자, 매개 변수 없고, 아무 일 없이 단순 리턴  
}
```

괄호도 비고 중괄호도 비었다.

생성자가 하나도 없으면 컴파일러가 몰래 넣는다

클래스에 생성자가 하나도 선언되어 있지 않은 경우, 컴파일러는 기본 생성자를 자동으로 생성한다.



기본 생성자로 동전 객체 만들기

Ex07_DefaultCtor.java

```
class Coin {  
    int radius;  
    void set(int r) { radius = r; }  
    double getArea() { return 3.14*radius*radius; }  
}  
  
public class Ex07_DefaultCtor {  
    public static void main(String[] args) {  
        Coin c = new Coin(); // 기본 생성자 Coin() 호출  
        c.set(5);  
        System.out.println(c.getArea());  
    }  
}
```

동전 넓이 결과

78.5

생성자를 안 썼는데도 객체가 만들어졌다

생성자를 하나 만들면 기본 생성자는 들어가지 않는다

생성자가 하나라도 존재하는 클래스에는 컴파일러가 기본 생성자를 임의로 삽입해 주지 않는다.

```
Circle pizza = new Circle(10); // Circle(int r) 호출
```

오류

```
Circle donut = new Circle(); // 컴파일 오류. 생성자 Circle() 없음
```

바퀴 클래스에 매개 변수 없는 new 를 쓰면

```
class Wheel {  
    int radius;  
    public Wheel(int r) {  
        radius = r;  
    }  
}
```

```
public class Ex08_NoDefaultCtor {  
    public static void main(String[] args) {  
        Wheel a = new Wheel(10);  
        Wheel b = new Wheel(); // 컴파일 오류. 생성자 Wheel() 없음  
        System.out.println(a.radius);  
    }  
}
```



컴파일 오류 원문

Ex08_NoDefaultCtor.java:11: error: constructor Wheel in class Wheel cannot be applied to given types;

Wheel b = new Wheel(); // 컴파일 오류. 생성자 Wheel() 없음

^

required: int

found: no arguments

reason: actual and formal argument lists differ in length

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

생성자를 하나 만든 클래스에는 Wheel() 이 없다

오류 줄의 required 와 found 읽기

줄

뜻

required: int

→ 있는 생성자는 int 하나를 받는다

found: no arguments

→ new Wheel() 은 아무것도 넘기지 않았다

reason: actual and formal argument lists differ in length

→ 넘긴 개수와 받는 개수가 다르다

required 는 있는 생성자가 받는 것, found 는 넘긴 것이다.

고치는 길 둘

Wheel 클래스를 고친다

생성자를 직접 하나 더 쓴다

```
public Wheel() { radius = 1; }
```

부르는 쪽을 고친다

new 에 값을 넣는다

```
Wheel b = new Wheel(20);
```

어느 쪽이든 부르는 new 와 있는 생성자가 맞으면 된다.

언제 기본 생성자가 생기나

클래스에 쓴 생성자

없음

Wheel(int r) 하나

Wheel() 과 Wheel(int r) 둘

new X() 는?

된다(컴파일러가 넣음)

오류

된다(직접 쓴 Wheel())

this - 객체 자신에 대한 레퍼런스

this는 객체 자신에 대한 레퍼런스로서 메소드 안에서 사용된다.

this 는 컴파일러가 알아서 관리한다. 우리는 쓰기만 한다.

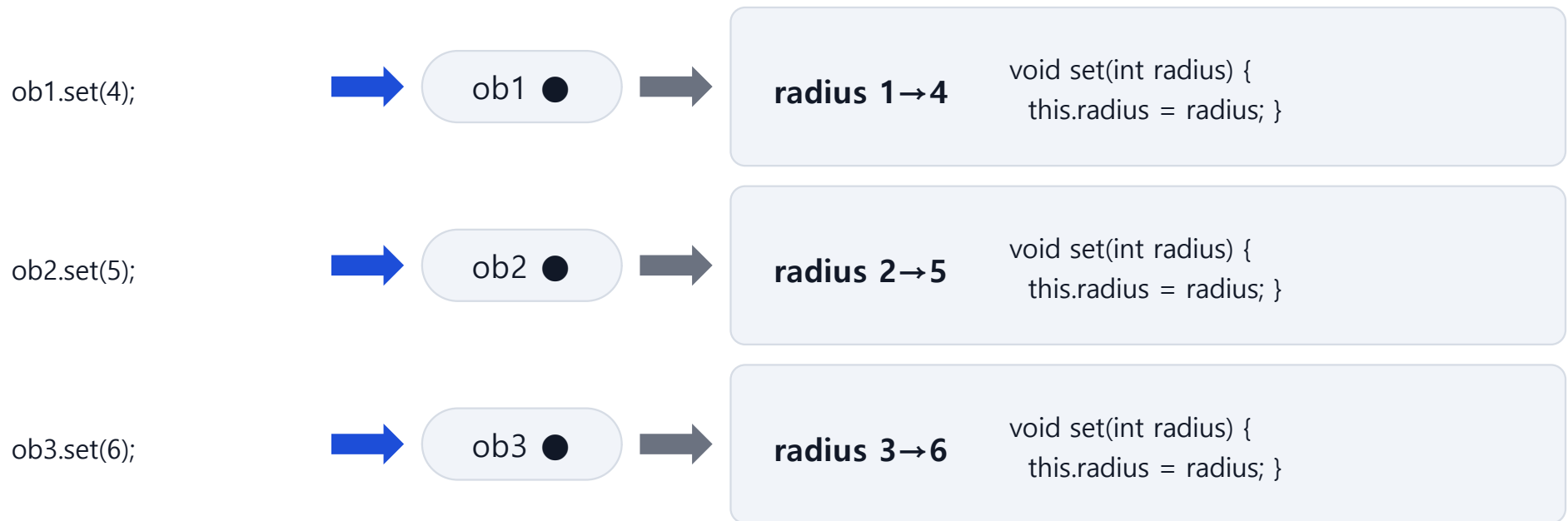
같은 이름이면 this 가 필드를 가리킨다

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

this.radius → 멤버 radius

오른쪽 radius → 매개 변수

객체마다 this 가 가리키는 것이 다르다



메소드 코드는 하나다. `this` 는 메소드를 부른 객체를 따라간다.

this 로 접시 반지름 바꾸기

Ex09_ThisSet.java

```
class Plate {  
    int radius;  
    public Plate(int radius) {  
        this.radius = radius;  
    }  
    void set(int radius) {  
        this.radius = radius;  
    }  
}
```

접시 세 개에 set 부르기

Ex09_ThisSet.java

```
public class Ex09_ThisSet {  
    public static void main(String[] args) {  
        Plate ob1 = new Plate(1);  
        Plate ob2 = new Plate(2);  
        Plate ob3 = new Plate(3);  
        ob1.set(4);  
        ob2.set(5);  
        ob3.set(6);  
        System.out.println(ob1.radius + " " + ob2.radius + " " + ob3.radius);  
    }  
}
```

접시 세 개 결과

4 5 6

세 객체가 저마다 자기 반지름을 바꿨다

this 를 두고 나눈 네 마디

1

this는 객체 자신에 대한 레퍼런스야.

2

this와 this()는 달라요.

3

this는 메소드에서 사용되며 현재 객체를 가리키죠.

4

하지만 static 메소드에서는 사용할 수 없어요.

this 를 빠뜨린 머그 용량 예측해 보기

Ex10_ThisMissing.java

결과를 먼저 예측한 뒤 실행해 비교한다.

```
class Mug {  
    int volume;  
    public Mug(int volume) {  
        volume = volume; // this 를 빠뜨렸다  
    }  
}  
  
public class Ex10_ThisMissing {  
    public static void main(String[] args) {  
        Mug m = new Mug(350);  
        System.out.println("머그 용량 " + m.volume + "ml");  
    }  
}
```

머그 용량이 0 으로 나온다

머그 용량 0ml

350 을 넣었는데 0 이 나왔다. 예측과 건줘 본다

volume = volume 은 무엇을 무엇에 넣었나

```
volume = volume;
```

매개 변수 volume 을
매개 변수 volume 에 넣었다

```
this.volume = volume;
```

매개 변수 volume 을
필드 volume 에 넣는다

이름이 같으면 안쪽(매개 변수)이 먼저 보인다.

this 로 학생 필드 채워 보기

Ex14_StudentThis.java

```
class Student {  
    String name;  
    int grade;  
    public Student(String name, int grade) {  
        this.name = name;  
        // 여기서 부터 코드를 작성하세요.  
  
        // 여기 까지  
    }  
}
```

생성자 안에서 grade 도 this 로 채운다.

학생 한 명 만들어 출력하기

Ex14_StudentThis.java

```
public class Ex14_StudentThis {  
    public static void main(String[] args) {  
        Student s = new Student("김자바", 1);  
        System.out.println(s.name + " " + s.grade + "학년");  
    }  
}
```

생성자 한 줄로 이름과 학년이 들어간다.

학생 결과

김자바 1학년

이름과 학년이 생성자 한 줄로 들어갔다

this()

생성자가 같은 클래스의 다른 생성자를 부르는 코드

this()는 클래스 내에서 생성자가 다른 생성자를 호출할 때 사용하는 자바 코드이다.

this → 객체 자신

this() → 생성자 호출

예제 4-5 this()로 다른 생성자 호출

Ex11_ThisCall.java

```
class Novel {  
    String title;  
    String author;  
  
    void show() { System.out.println(title + " " + author); }  
  
    public Novel() {  
        this("", "");  
        System.out.println("생성자 호출됨");  
    }  
}
```

8행 this("", "") 가 매개 변수 둘인 생성자를 부른다.

예제 4-5 매개 변수 둘인 생성자가 일을 맡는다

Ex11_ThisCall.java

필드를 채우는 곳은 이 생성자 한 곳이다.

```
public Novel(String title) {  
    this(title, "작자미상");  
}  
  
public Novel(String title, String author) {  
    this.title = title;  
    this.author = author;  
}  
}
```

예제 4-5 책 세 권 만들기

Ex11_ThisCall.java

```
public class Ex11_ThisCall {  
    public static void main(String[] args) {  
        Novel javaBook = new Novel("Java", "황기태");  
        Novel bible = new Novel("Bible");  
        Novel emptyBook = new Novel();  
  
        bible.show();  
    }  
}
```

예제 4-5 실행 결과

생성자 호출됨
Bible 작자미상

빈 책을 만들 때 생성자 호출됨 이 찍혔다

new Novel("Bible") 이 거쳐 가는 길

1

main 에서 부른다
new Novel("Bible")



2

매개 변수 하나인 생성자가 this() 를 실행한다
Novel(String title)
this(title, "작자미상");



3

매개 변수 둘인 생성자가 필드를 채운다
Novel(String title, String author)

채워진 필드

title = "Bible"
author = "작자미상"

this() 를 쓸 때 지킬 세 가지

1

this()는 생성자에서만 사용된다.

2

this()는 반드시 같은 클래스의 다른 생성자를 호출할 때 사용된다.

3

this()는 반드시 생성자의 첫 번째 문장으로 사용되어야 한다.

this() 를 두 번째 줄에 쓰면

교수 시연

Ex12_ThisFirst.java



```
class Diary {  
    String title;  
    public Diary() {  
        System.out.println("생성자 호출됨");  
        this(""); // 컴파일 오류. 첫 번째 문장이 아니다  
    }  
    public Diary(String title) { this.title = title; }  
}  
  
public class Ex12_ThisFirst {  
    public static void main(String[] args) {  
        Diary d = new Diary();  
    }  
}
```

컴파일 오류 원문

Ex12_ThisFirst.java:5: error: call to this must be first statement in constructor

```
this(""); // 컴파일 오류. 첫 번째 문장이 아니다
```

^

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

this() 는 생성자의 첫 문장이어야 한다

생성자 셋이 차례로 불린다

Ex13_CtorOrder.java

결과를 먼저 예측한 뒤 실행해 비교한다.

```
class Cake {  
    String flavor;  
    int size;  
    public Cake() {  
        this("바닐라");  
        System.out.println("3. Cake() 끝");  
    }  
    public Cake(String flavor) {  
        this(flavor, 1);  
        System.out.println("2. Cake(String) 끝");  
    }  
}
```

케이크 하나 만들 때 찍히는 순서 예측해 보기

Ex13_CtorOrder.java

```
public Cake(String flavor, int size) {  
    this.flavor = flavor; this.size = size;  
    System.out.println("1. Cake(String, int) 끝");  
}  
  
public class Ex13_CtorOrder {  
    public static void main(String[] args) {  
        Cake c = new Cake();  
        System.out.println(c.flavor + " " + c.size + "호");  
    }  
}
```

생성자 호출 순서 결과

1. Cake(String, int) 끝
 2. Cake(String) 끝
 3. Cake() 끝
- 바닐라 1호

맨 안쪽 생성자부터 끝난다. 예측한 순서와 견줘 본다

this() 로 기본 용량 정해 보기

직접 실습

Ex15_CupThis.java

음료 이름만 받는 생성자에서 this() 로 용량 355 를 넘긴다.

```
class Cup {
    String drink;
    int ml;
    public Cup(String drink) {
        // 여기서 부터 코드를 작성하세요.

        // 여기 까지
    }
    public Cup(String drink, int ml) {
        this.drink = drink;
        this.ml = ml;
    }
}
```

컵 두 개 만들어 출력하기

Ex15_CupThis.java

```
public class Ex15_CupThis {  
    public static void main(String[] args) {  
        Cup a = new Cup("라떼", 473);  
        Cup b = new Cup("아메리카노");  
        System.out.println(a.drink + " " + a.ml + "ml");  
        System.out.println(b.drink + " " + b.ml + "ml");  
    }  
}
```

컵 두 개 결과

라떼 473ml

아메리카노 355ml

용량을 안 넣은 컵은 this() 가 정한 355ml

문제: Samp 클래스의 잘못된 세 군데

다음 코드에서 잘못된 부분이 3군데 있다. 모두 수정하라.

```
class Samp {  
    int id;  
    public void Samp(int x) { this.id = x; }  
    public void Samp() {  
        System.out.println("생성자 호출");  
        this(0);  
    }  
}
```

풀이: void 두 개와 this(0) 자리

```
class Samp {  
    int id;  
    public Samp(int x) { this.id = x; }  
    public Samp() {  
        this(0);  
        System.out.println("생성자 호출");  
    }  
}  
  
public class Ex16_SampFix {  
    public static void main(String[] args) {  
        Samp a = new Samp();  
        System.out.println("id = " + a.id);  
    }  
}
```

Ex16_SampFix.java

고칠 곳 1

생성자 앞의 void 를 지운다.

고칠 곳 2

매개 변수 없는 생성자 앞의 void 도 지운다.

고칠 곳 3

this(0) 을 생성자의 첫 문장으로 올린다.

고친 Samp 실행 결과

생성자 호출
id = 0

this(0) 이 먼저 돌고 출력 줄이 그다음에 돈다

2교시에서 본 것

기본 생성자

생성자를 하나도 안 쓰면 컴파일러가 기본 생성자를 넣는다.

생성자를 하나라도 쓰면 기본 생성자는 들어가지 않는다.

this · this()

this 는 지금 일하는 객체 자신을 가리킨다.

this() 는 생성자의 첫 문장에서 다른 생성자를 부른다.

2교시 실습 살펴보기

오류 없이 틀리는 실수

`volume = volume`

`volume = volume` 처럼 오류 없이 결과만 0 이 되는 실수는 실행해 봐야 보인다.

생성자 호출 순서

생성자가 이어질 때는 맨 안쪽부터 끝난다
. 출력 순서로 확인한다.

3교시

인자 전달과 메소드 오버로딩

메소드에 무엇이 넘어가는지 보고 같은 이름의 메소드를 여러 개 만든다.

값에 의한 호출

메소드 오버로딩

객체 치환

메소드는 이름표 다섯 개로 읽는다

```
public int getSum(int i, int j) {  
    int sum;  
    sum = i + j;  
    return sum;  
}
```

접근 지정자

public

리턴 타입

int

메소드 이름

getSum

메소드 인자들

int i, int j

메소드 코드

중괄호 안

리턴 타입과 return 은 5주차에 봤다. 오늘은 괄호 안의 인자를 본다.

예제의 static 은 main 에서 바로 부르려고 붙였다

static 은 main 에서 객체 없이 바로 부르려고 붙인 말이다.
뜻은 7주차에 배운다.

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

값에 의한 호출

실인자의 값이 복사되어 매개 변수에 전달된다

호출 방식

자바의 메소드 호출 시 인자 전달 방식은 '값에 의한 호출'(call-by-value)이다.

값 전달

호출하는 실인자의 값이 복사되어 메소드의 매개 변수에 전달된다.

기본 타입을 넘기면 값만 복사된다

Ex17_CallByValue.java

```
public class Ex17_CallByValue {  
    public static void main(String[] args) {  
        int n = 10;  
  
        increase(n);  
  
        System.out.println(n);  
    }  
  
    static void increase(int m) {  
        m = m + 1;  
    }  
}
```

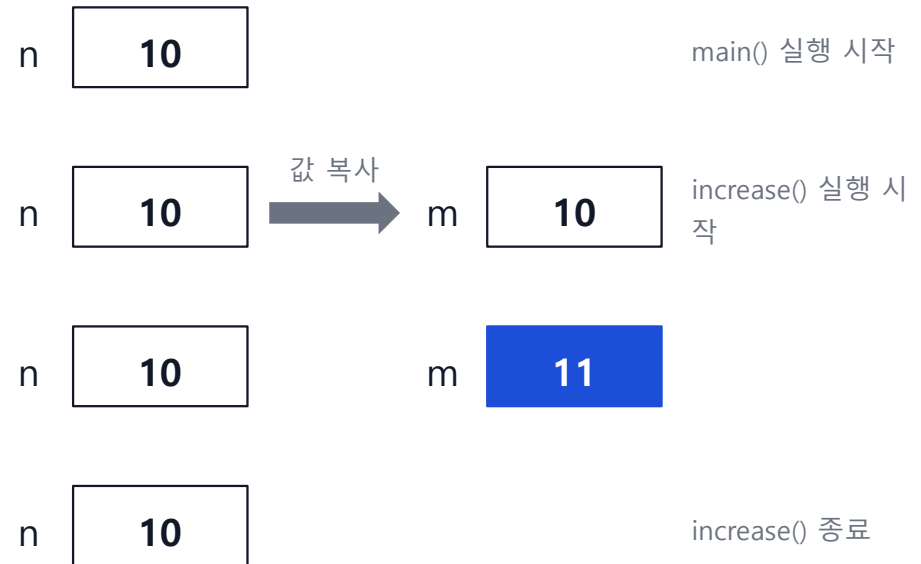
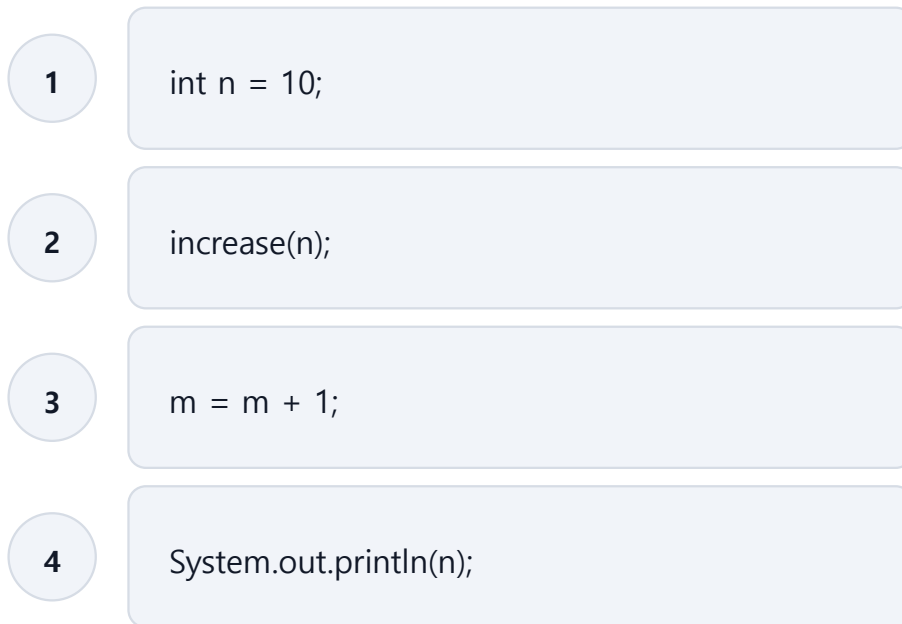


n 은 그대로 10

10

increase 가 m 을 11 로 바꿨지만 n 은 그대로 10

n 과 m 은 다른 칸이다



복사본 m 을 고쳐도 원본 n 은 바뀌지 않는다.

객체를 넘기면 레퍼런스가 복사된다

Ex18_ReferencePassing.java

```
class Balloon {  
    int radius;  
    public Balloon(int radius) {  
        this.radius = radius;  
    }  
}
```

풍선 반지름을 메소드 안에서 늘리기

Ex18_ReferencePassing.java

```
public class Ex18_ReferencePassing {  
    public static void main(String[] args) {  
        Balloon pizza = new Balloon(10);  
  
        increase(pizza);  
  
        System.out.println(pizza.radius);  
    }  
  
    static void increase(Balloon m) {  
        m.radius++;  
    }  
}
```

반지름은 11

11

메소드 안에서 늘린 반지름이 main 에서도 11

pizza 와 m 은 같은 객체를 가리킨다

1 pizza = new Balloon(10);



2 increase(pizza);



3 m.radius++;



4 System.out.println(pizza.radius);



복사된 것은 화살표이고 객체는 하나뿐이다.

배열을 넘겨도 레퍼런스만 간다

Ex19_ArrayPassing.java

```
public class Ex19_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]+" ");  
    }  
}
```

increase 가 배열 원소를 하나씩 늘린다

Ex19_ArrayPassing.java

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

increase 는 받은 배열의 원소를 모두 1 씩 늘린다.

2 3 4 5 6

2 3 4 5 6

메소드 안에서 늘린 원소가 main 에도 남아 2 3 4 5 6

잠깐: 레퍼런스 전달의 좋은 점과 주의할 점

- 자바에서는 메소드의 매개 변수로 객체나 배열을 전달할 때 레퍼런스만 전달하기 때문에, 객체나 배열이 통째로 넘어가지 않는다.
- 아무리 큰 객체나 배열도 하나의 정수 크기인 레퍼런스, 즉 일종의 주소만 전달되므로, 매개 변수 전달로 인한 시간이나 메모리의 오버헤드가 없는 장점이 있다.
- 하지만 메소드가 전달받은 객체의 필드나 배열의 원소 값을 변경할 수 있기 때문에 소위 부작용(side-effect)이 발생할 수 있으므로 주의해야 한다.

좋은 점: 큰 배열도 주소만 넘어간다

주의할 점: 메소드가 원본을 바꿀 수 있다

예제 4-8 공백을 쉼표로 바꾸는 메소드

Ex20_CharReplace.java

```
public class Ex20_CharReplace {  
    static void replaceSpace(char a[]) { // 배열 a의 공백문자를 ';'로 변경  
        for(int i = 0; i < a.length; i++)  
            if(a[i] == ' ') // 공백 문자를 ';'로 변경  
                a[i] = ';'  
    }  
  
    static void printCharArray(char a[]) { // 배열 a의 문자들을 화면에 출력  
        for(int i = 0; i < a.length; i++)  
            System.out.print(a[i]); // 배열 원소 문자 출력  
        System.out.println(); // 배열 원소 모두 출력 후 줄바꿈  
    }  
}
```

예제 4-8 바꾸기 전과 후 출력

Ex20_CharReplace.java

```
public static void main(String[] args) {  
    char c[] = {'T','h','i','s',' ','i','s',' ','a',' ','p','e','n','c','i','l','.'};  
    printCharArray(c); // 원래 배열 원소 출력  
    replaceSpace(c); // 공백 문자 바꾸기  
    printCharArray(c); // 수정된 배열 원소 출력  
}  
}
```

예제 4-8 실행 결과

This is a pencil.

This,is,a,pencil.

메소드가 원본 배열의 공백을 쉼표로 바꿨다

☕ 같은 char 배열을 println 이 두 번 다르게 찍는다

```
char c[] = {'J','a','v','a'};  
System.out.println(c);      // Java  
System.out.println("" + c); // [C@2a139a55 (뒤 숫자는 실행마다 다르다)
```

[C@ 뒤의 숫자는 실행할 때마다 다르다.



println 은 넘긴 값의 타입을 보고 고른다

넘긴 값

char 배열 c

문자열 "" + c

불리는 println

`println(char[])`

글자를 찍는다

`println(String)`

그 문자열을 찍는다

이름이 같아도 받는 타입이 다르면 서로 다른 메소드다.

swap 으로 두 수가 바뀔지 예측해 보기

```
public class Ex21_SwapTry {  
    static void swap(int x, int y) {  
        int t = x;  
        x = y;  
        y = t;  
        System.out.println("swap 안: " + x + " " + y);  
    }  
  
    public static void main(String[] args) {  
        int a = 3, b = 7;  
        swap(a, b);  
        System.out.println("main: " + a + " " + b);  
    }  
}
```

Ex21_SwapTry.java

할 일

결과를 먼저 예측한 뒤
실행해 비교한다.

main 의 a 와 b 는 그대로

swap 안: 7 3
main: 3 7

swap 안에서는 바뀌었지만 main 의 a 와 b 는 3 7 그대로

점수 배열에 보너스 더해 보기

직접 실습

Ex22_BonusArray.java

할 일: addBonus 안에서 score 배열의 원소마다 bonus 를 더한다.

```
public class Ex22_BonusArray {  
    static void addBonus(int[] score, int bonus) {  
        // 여기서 부터 코드를 작성하세요.  
  
        // 여기 까지  
    }  
  
    public static void main(String[] args) {  
        int[] score = {70, 85, 90};  
        addBonus(score, 5);  
        for(int s : score)  
            System.out.print(s + " ");  
    }  
}
```

보너스 결과

75 90 95

세 점수에 5점씩 더해져 75 90 95

메소드 오버로딩

이름은 같고 매개 변수의 개수나 타입이 다른 메소드들

한 클래스 안에 이름이 같지만 매개 변수의 타입이나 개수가 서로 다른 여러 개의 메소드를 작성할 수 있다.

메소드 중복

다형성의 한 종류

오버로딩이 되려면 두 조건을 모두 맞춘다

조건 1

메소드 이름이 동일하여야 한다.



조건 2

메소드 매개 변수의 개수나 타입이 서로 달라야 한다.

판단과 무관

메소드의 리턴 타입은 오버로딩의 성공을 판단하는 것과 무관하며, 접근 지정자 역시 메소드 오버로딩과 관계없다.

getSum 세 개가 한 클래스에 산다

Ex23_MethodSample.java

(int, int)

(int, int, int)

(double, double)

```
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;  
    }  
}
```

인자를 보고 컴파일러가 하나를 고른다

Ex23_MethodSample.java

```
public class Ex23_MethodSample {  
    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(1.5, 2.5);  
        System.out.println(i + " " + j + " " + k);  
    }  
}
```

넘긴 인자의 개수와 타입으로 어느 getSum 을 부를지 정해진다

3 6 4.0

3 6 4.0

인자 둘, 인자 셋, double 둘에 맞는 getSum 이 각각 불렀다

리턴 타입만 다르면 오버로딩이 아니다

```
class MethodOverloadingFail { // 메소드 오버로딩이 실패한 사례
    public int getSum(int i, int j) {
        return i + j;
    }
    public double getSum(int i, int j) {
        return (double)(i + j);
    }
}

public class Ex24_OverloadFail {
    public static void main(String[] args) {
        MethodOverloadingFail f = new MethodOverloadingFail();
        System.out.println(f.getSum(1, 2));
    }
}
```

Ex24_OverloadFail.java

매개 변수의 개수와 타입이 같기 때문에 오버로딩 실패, 이 두 메소드는 같은 클래스에 공존할 수 없습니다.

컴파일 오류 원문

Ex24_OverloadFail.java:5: error: method getSum(int,int) is already defined in class MethodOverloadingFail

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

^

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

리턴 타입만 다른 getSum 은 이미 있는 메소드로 본다

이름 하나로 세 가지 넓이 구하기

Ex25_AreaOverload.java

```
class Calc {  
    int area(int side) {  
        return side * side;  
    }  
    int area(int w, int h) {  
        return w * h;  
    }  
    double area(double r) {  
        return 3.14 * r * r;  
    }  
}
```

정사각형, 직사각형, 원 넓이 출력하기

Ex25_AreaOverload.java

```
public class Ex25_AreaOverload {  
    public static void main(String[] args) {  
        Calc c = new Calc();  
        System.out.println(c.area(4));  
        System.out.println(c.area(3, 5));  
        System.out.println(c.area(2.0));  
    }  
}
```

넓이 세 개 결과

16

15

12.56

인자의 개수와 타입으로 세 area 가 같았다

어느 쪽인지 정할 수 없는 호출

교수 시연

Ex26_Ambiguous.java

```
class Printer {
    void show(int a, double b) {
        System.out.println("int, double");
    }
    void show(double a, int b) {
        System.out.println("double, int");
    }
}

public class Ex26_Ambiguous {
    public static void main(String[] args) {
        Printer p = new Printer();
        p.show(1, 2); // 컴파일 오류. 어느 쪽인지 정할 수 없다
    }
}
```

컴파일 오류 원문

Ex26_Ambiguous.java:13: error: reference to show is ambiguous

p.show(1, 2); // 컴파일 오류. 어느 쪽인지 정할 수 없다

^

both method show(int,double) in Printer and method show(double,int) in Printer match

1 error

컴파일에서 막혀 실행하지 못했어요. 위의 오류를 먼저 고쳐 주세요.

두 show 가 똑같이 맞아 하나로 정할 수 없다

생성자도 오버로딩된다

예제 4-3 Circle 생성자

```
public Circle()
```

```
public Circle(int r, String n)
```



이름이 같다(Circle)



매개 변수가 다르다
(없음 · int 와 String)

1교시 생성자 여러 개, 2교시 this() 가 모두 오버로딩이었다.

☕ null 을 넘기면 어느 생성자가 불릴까

```
Confusing(Object o)      → "Object"  
Confusing(double[] dArray) → "double array"  
new Confusing(null);    // double array
```



더 구체적인 쪽이 골라진다

넓은 타입

Object 를 받는 쪽: 모든 객체가 들어간다

좁은 타입

double[] 를 받는 쪽: double 배열만 들어간다

둘 다 맞으면 좁은 쪽이 골라지고, 넓고 좁음을 가릴 수 없으면 앞에서 본 모호성 오류가 난다.

max 를 세 수 판으로 오버로딩해 보기

Ex27_MaxOverload.java

할 일: 세 수 중 가장 큰 값을 m 에 넣는다. 이미 있는 max(a, b) 를 불러도 된다.

```
class Picker {  
    int max(int a, int b) {  
        return a > b ? a : b;  
    }  
    int max(int a, int b, int c) {  
        int m = 0;  
        // 여기서 부터 코드를 작성하세요.  
  
        // 여기 까지  
        return m;  
    }  
}
```

두 수와 세 수의 최댓값 출력하기

Ex27_MaxOverload.java

```
public class Ex27_MaxOverload {  
    public static void main(String[] args) {  
        Picker p = new Picker();  
        System.out.println(p.max(3, 8));  
        System.out.println(p.max(3, 8, 5));  
    }  
}
```

p.max(3, 8) 은 인자 둘, p.max(3, 8, 5) 는 인자 셋으로 max 를 부른다.

최댓값 결과

8

8

두 수에서도 세 수에서도 가장 큰 값은 8

객체 치환은 객체를 복사하지 않는다

객체의 치환은 객체를 복사하는
것이 아니다.

`ob1 = ob2;` 는 `ob2` 가 가리키는 객체를 `ob1` 도
가리키게 한다.

5주차 Point 공유(이름 둘, 객체 하나)



배지 두 개를 만드는 Badge 클래스

Ex28_SampAssign.java

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

생성자 하나와 set · get

ob1 = ob2 다음에 id 출력하기

Ex28_SampAssign.java

```
public class Ex28_SampAssign {  
    public static void main(String[] args) {  
        Badge ob1 = new Badge(3);  
        Badge ob2 = new Badge(4);  
        Badge s;  
  
        s = ob2;  
        ob1 = ob2; // 객체의 치환  
        System.out.println("ob1.id="+ob1.get());  
        System.out.println("ob2.id="+ob2.get());  
    }  
}
```

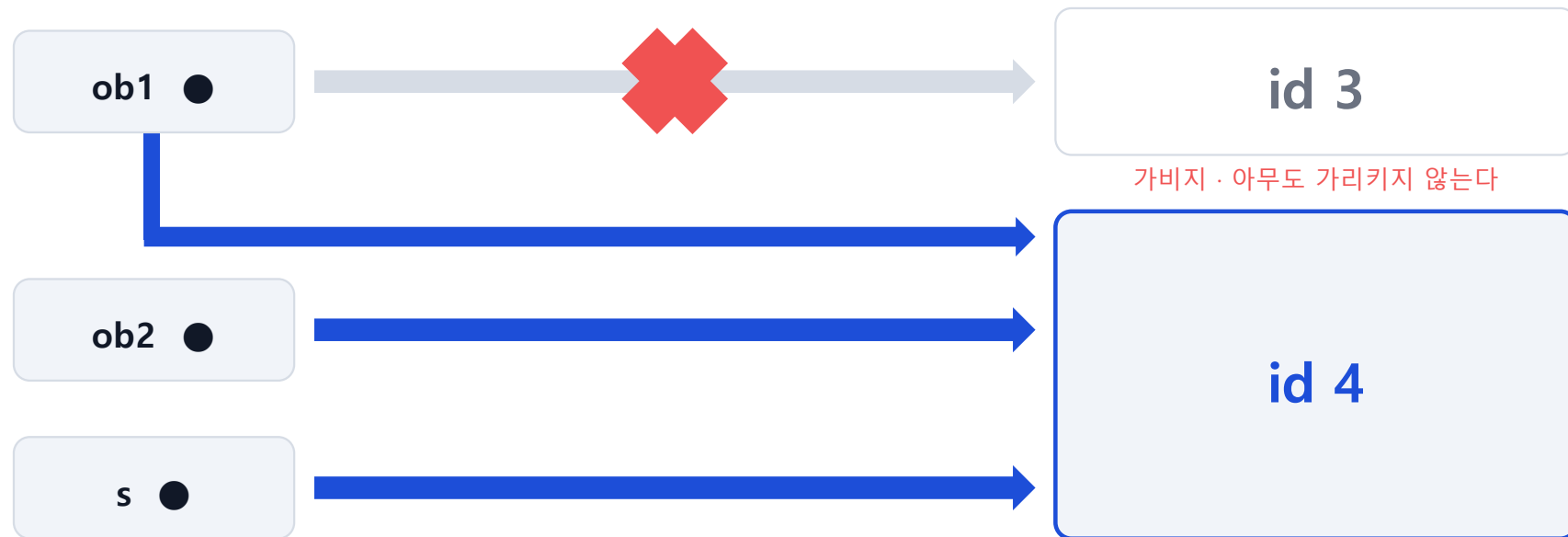
둘 다 id=4

ob1.id=4

ob2.id=4

ob1 도 ob2 의 객체를 가리켜 둘 다 id=4

ob1 이 가리키던 객체는 갈 곳이 없어진다



아무도 가리키지 않는 객체를 가비지라고 부른다. 자세한 것은 7주차다.

생성자 둘을 가진 Ticket

Ex29_TicketTotal.java

```
class Ticket {  
    String movie;  
    int count;  
    public Ticket(String movie) {  
        this(movie, 1);  
    }  
    public Ticket(String movie, int count) {  
        this.movie = movie;  
        this.count = count;  
    }  
}
```

할 일: 할인 금액을 받는 price 의 빈칸을 채워 표 두 장의 금액을 출력한다.

할인 금액을 받는 price 채워 보기

Ex29_TicketTotal.java

```
int price() { return count * 12000; }  
int price(int discount) {  
    int total = 0;  
    // 여기서 부터 코드를 작성하세요.  
  
    // 여기 까지  
    return total;  
}  
}
```

discount 는 한 장당 할인 금액, count 는 장수

표 두 장의 금액 출력하기

Ex29_TicketTotal.java

```
public class Ex29_TicketTotal {  
    public static void main(String[] args) {  
        Ticket a = new Ticket("인사이드 아웃");  
        Ticket b = new Ticket("주토피아", 3);  
        System.out.println(a.movie + " " + a.price() + "원");  
        System.out.println(b.movie + " " + b.price(2000) + "원");  
    }  
}
```

인사이드 아웃은 한 장, 주토피아는 세 장으로 만든다.

표 두 장 결과

인사이드 아웃 12000원
주토피아 30000원

한 장은 12000원, 세 장에서 장당 2000원을 빼면 30000원

3교시에서 본 것

- 1 기본 타입을 넘기면 값이 복사된다
- 2 객체와 배열을 넘기면 레퍼런스가 복사된다
- 3 이름이 같고 매개 변수가 다르면 오버로딩이다
- 4 리턴 타입만 다르면 오버로딩이 아니다
- 5 객체 치환은 레퍼런스를 옮길 뿐 객체를 복사하지 않는다

3교시 실습 살펴보기

1

swap 은 메소드 안에서만 바뀌고, 배열 보너스는 main 에도 남는다. 넘긴 것이 값인지 레퍼런스인지가 갈랐다(실습 ⑦ · ⑧)

2

max 와 price 는 같은 이름의 다른 메소드를 안에서 불러 다시 썼다(실습 ⑨ · ⑩)

내 연락처 클래스 만들기

미션 조건

- 생성자: 이름만 · 이름과 번호
- this(): 짧은 쪽이 긴 쪽을 부른다
- 제출: 없음 (자율 학습)

혼자 해 볼 때 볼 세 가지

생성자 이름이 클래스 이름과 같은가
매개 변수와 필드 이름이 같을 때 this 를 붙였나
this() 가 생성자의 첫 문장인가

안 될 때 여기부터

틀림

- 생성자를 만든 뒤 new 클래스()
- 생성자 둘째 줄에 this()
- 매개 변수 이름 = 매개 변수 이름

맞음

- ✓ 매개 변수 없는 생성자를 만든다
- ✓ this() 를 맨 위로 옮긴다
- ✓ this.필드 = 매개 변수

오류 줄의 required 와 found 를 먼저 읽는다

오늘 공부한 내용

생성자는 객체가 생성될 때 초기화를 위해 실행되는 메소드
생성자가 하나도 없을 때만 컴파일러가 기본 생성자를 넣는다
this 는 객체 자신, this() 는 같은 클래스의 다른 생성자
기본 타입은 값이, 객체와 배열은 레퍼런스가 복사된다
오버로딩은 이름이 같고 매개 변수의 개수나 타입이 다르다

다음 주엔 가비지 컬렉션부터

객체가 사라지는 때와 멤버를 감추는 법을 봅니다



다음 주 예고

- 가비지 컬렉션
- 접근 지정자와 static