

생각하기

■ TV와 관련된 명사와 동사는 무엇인가요?

- 명사(변수 또는 상수) 2개 이상
- 동사(메서드) 2개 이상

■ 통장과 관련된 명사와 동사는 무엇인가요?

- 명사(변수 또는 상수) 2개 이상
- 동사(메서드) 2개 이상

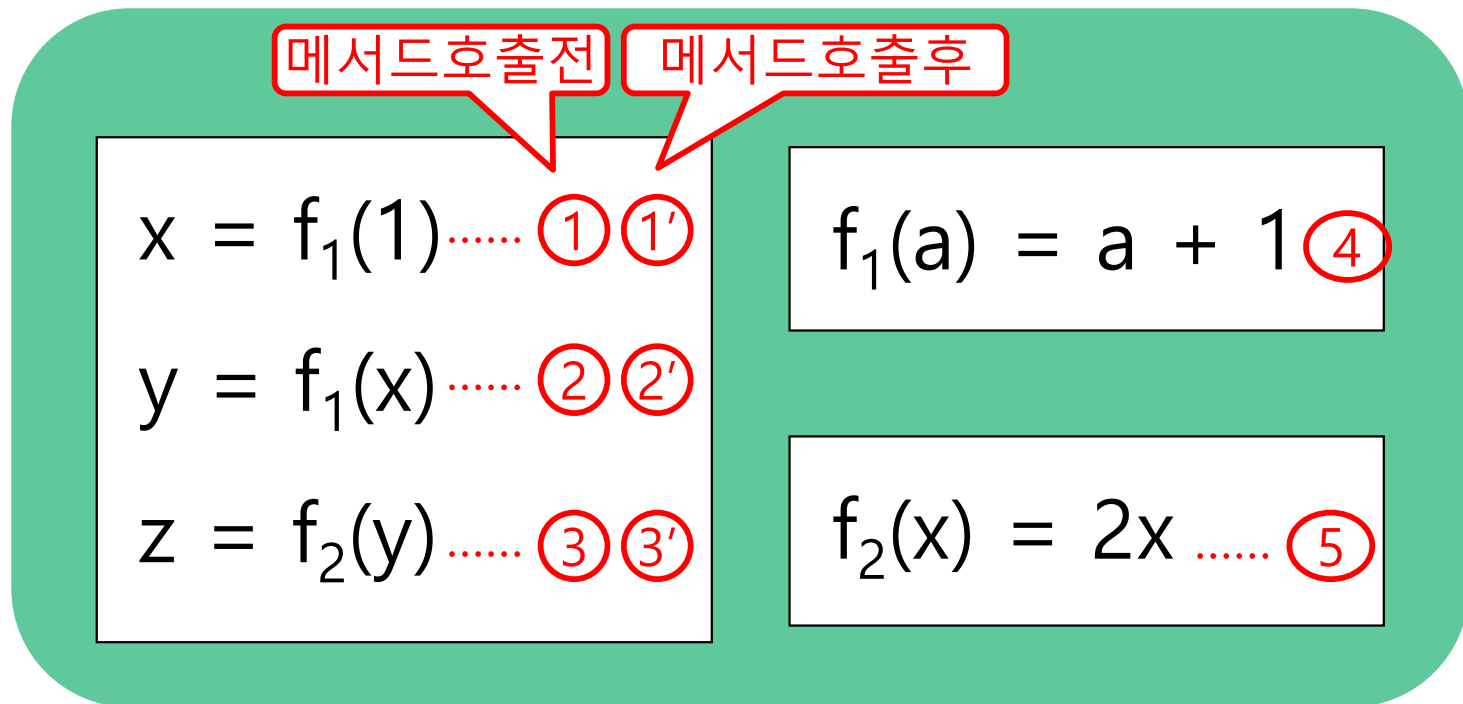
■ 가위바위보와 관련된 명사와 동사는 무엇인가요?

- 명사(변수 또는 상수) 2개 이상
- 동사(메서드) 2개 이상

생각하기

■ 메서드의 이해

- 메서드 x , y , z 의 값은? 계산되는 순서는? x 의 영향력?



생각하기

■ !를 계산하는 알고리즘 작성하시오

- n=0일 경우를 고려하여 괄호에 적당한 수식을 작성

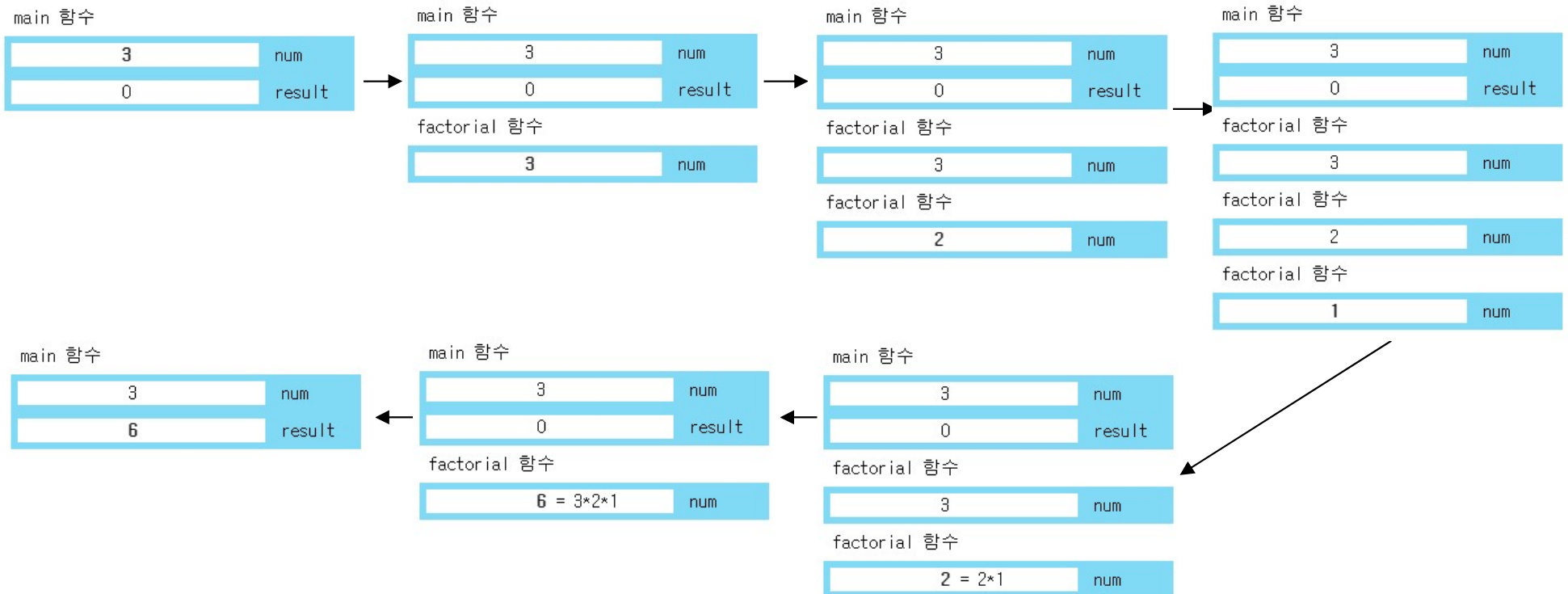
$$n! = n \times (\quad)!$$

$$\sum_{i=0}^n i = n + \sum_{i=0}^{(\quad)} i$$

factorial ! 계산 프로그램

```
public int factorial( int num )  
{  
    ...  
    num *= factorial( num - 1 );  
    ...  
}
```

숫자를 입력하세요: 3
 $3! = 3 * 2 * 1 = 6$



생각하기

■ 하노이 타워 알고리즘을 작성하시오

- 기둥이 3개 있고 원판이 N 개 있을 때 왼쪽기둥에서 오른쪽 기둥으로 원판을 모두 옮기세요. 단 원판은 자기보다 작은 원판 위에 놓을 수 없습니다.



생각하기



Original Configuration



Fourth Move



First Move



Fifth Move



Second Move



Sixth Move



Third Move



Seventh and Last Move



클래스와 객체

세상 모든 것이 객체

9

▣ 세상 모든 것이 객체



TV



의자



책



집



카메라



컴퓨터

▣ 실세계 객체의 특징

- 객체마다 고유한 특성(state)과 행동(behavior)를 가짐
- 다른 객체들과 정보를 주고 받는 등, 상호작용하면서 존재

▣ 컴퓨터 프로그램에서 객체 사례

- 테트리스 게임의 각 블록들
- 한글 프로그램의 메뉴나 버튼들

클래스와 객체

10

□ 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: 봉어빵틀, 객체: 구워낸 봉어빵들

사람을 사례로 든 클래스와 객체 사례

11

클래스: 사람

이름, 직업, 나이, 성별, 혈액형
밥 먹기, 잠자기, 말하기, 걷기



이름 황수연
직업 컴퓨터과학 전공 학생
나이 20
성별 여
혈액형 A

객체: 황수연



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

12

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

13

- 클래스 선언, class Circle
 - ▣ class 키워드로 선언
 - ▣ 클래스는 { } 사이에 모든 필드와 메소드 구현
- 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
- 생성자
 - ▣ 클래스의 이름과 동일한 특별한 메소드
 - ▣ 객체가 생성될 때 자동으로 한 번 호출되는 메소드
 - ▣ 개발자는 객체를 초기화하는데 필요한 코드 작성

객체 생성과 접근

1. 레퍼런스 변수 선언

(1) Circle pizza;

pizza




2. 객체 생성

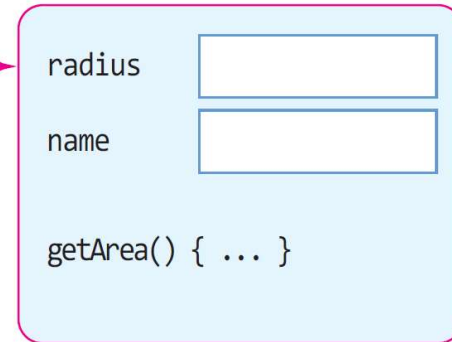
- new 연산자 이용

(2) pizza = new Circle();

pizza



Circle 타입의 객체



객체 메모리 할당 및 객체 생성

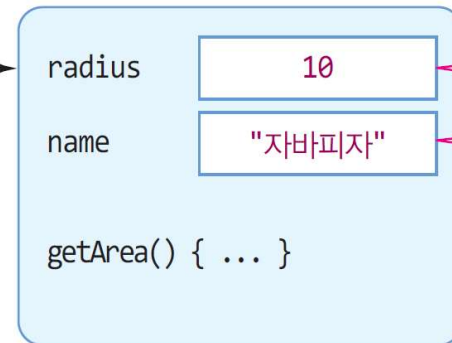
3(4). 객체 멤버 접근

- 점(.) 연산자 이용

(3) pizza.radius = 10;

pizza.name = "자바피자"

pizza



radius 값 변경

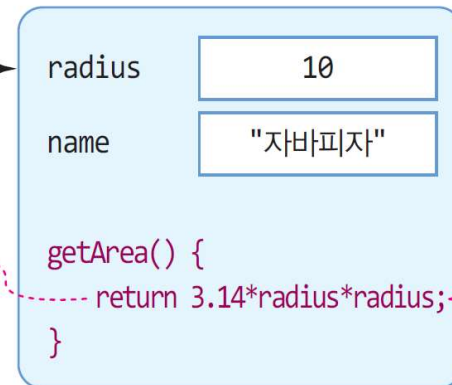
name 값 변경

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

pizza



area



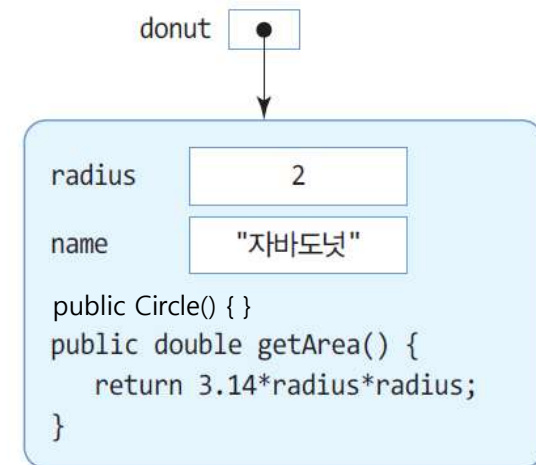
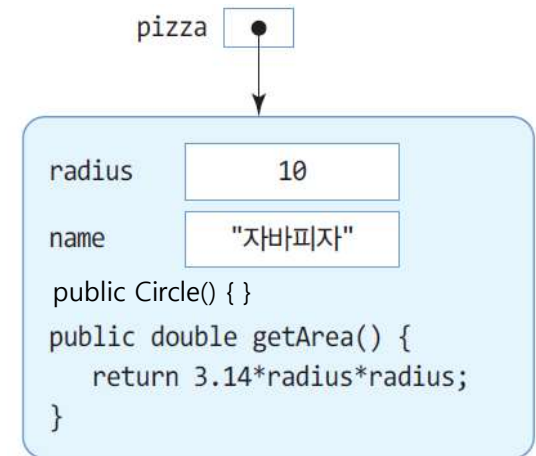
getArea() 메소드 실행

예제 4-1 : Circle 클래스의 객체 생성 및 활용

15

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

생성자 개념

16

□ 생성자

- ▣ 객체가 생성될 때 초기화를 위해 실행되는 메소드

□ 생성자의 특징

- ▣ 생성자 이름은 클래스 이름과 반드시 동일하고 리턴 타입 없음
- ▣ new에서 객체를 생성할 때 호출하며, 여러 개 작성 가능(오버로딩)
- ▣ 개발자가 생성자 미작성시 컴파일러가 자동으로 기본 생성자 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
생성자

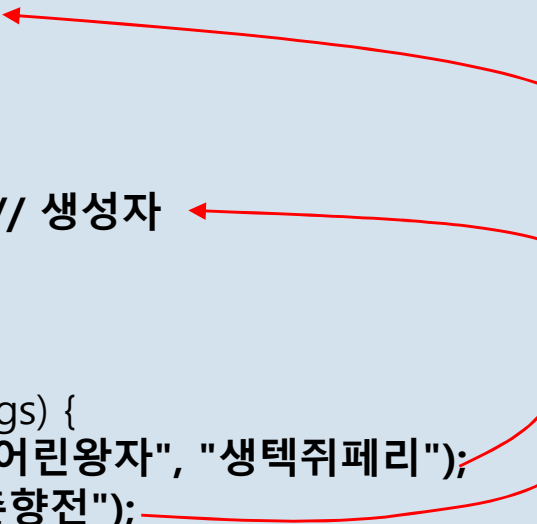
컴파일러가 자동으로 기본 생성자 삽입

예제 4-4 : 생성자 선언 및 활용 연습

17

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 생성자가 자동 생성되지 않는 경우

18

- 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; }  
}
```

컴파일러가 기본 생성자를 자동 생성하지 않음
public Circle() { }

```
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());  
}
```

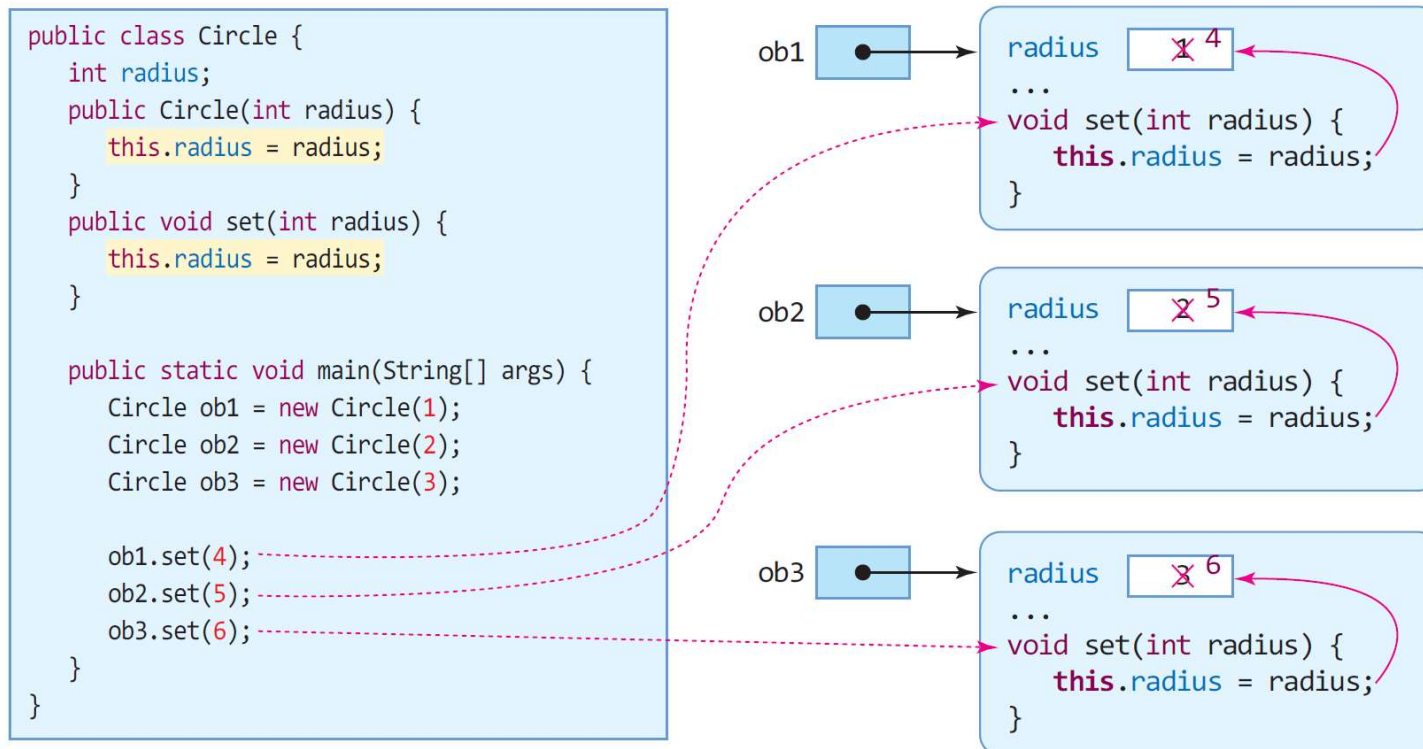
컴파일 오류.
해당하는 생성자가 없음 !!!

오류

this 레퍼런스

19

- ▣ 객체 자신에 대한 레퍼런스
 - 컴파일러에 의해 자동 관리, 개발자는 사용하기만 하면 됨
 - **this.멤버** 형태로 멤버 사용
- ▣ this의 필요성
 - 객체의 멤버 변수와 메소드 변수의 이름이 같은 경우
 - 메소드가 객체 자신의 레퍼런스를 반환할 때



this()로 다른 생성자 호출

20

□ this()

- 생성자에서 클래스 내의 다른 생성자 호출 가능
- 반드시 생성자 코드의 제일 처음에 수행

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

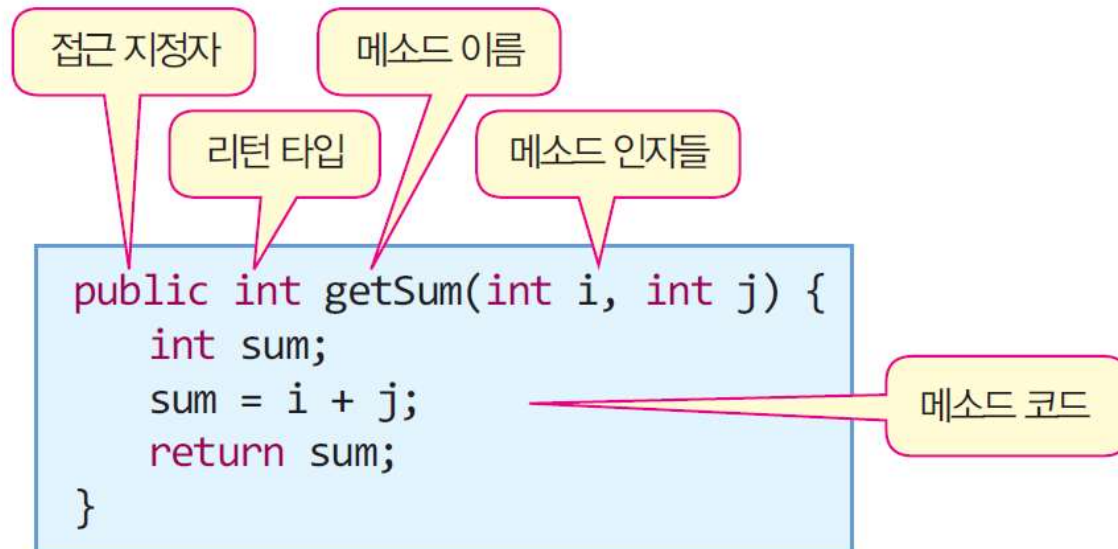
title = " 춘향전"
author = "작자미상"

생성자 호출됨
춘향전 작자미상

메소드 형식

21

- 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



인자 전달

22

□ 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

10 m

increase(int m) 실행 시작

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.radius++;

increase(Circle m) 종료

System.out.println(pizza.radius);

pizza



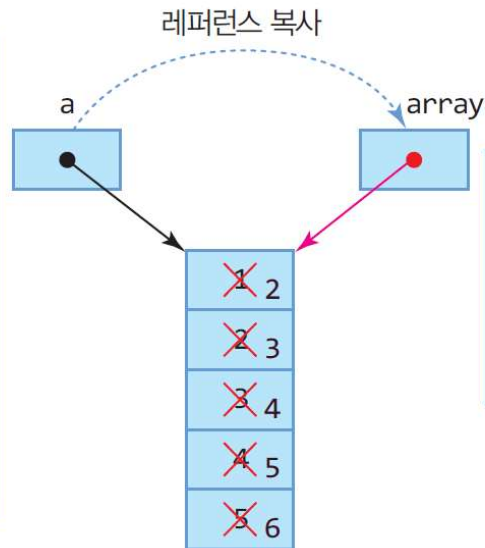
인자 전달 - 배열이 전달되는 경우

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

→ 실행 결과

2 3 4 5 6



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

메소드 오버로딩

26

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

// 메소드 오버로딩이 성공한 사례

```
private String selectRockPaperScissors( Scanner scan ) {  
    System.out.print( "가위, 바위, 보 중 하나를 입력하세요 : ");  
    return scan.next();  
}  
  
private String selectRockPaperScissors() {  
    int num = (int)(Math.random()*3);  
    if ( num == SCISSORS )  
        return "가위";  
    else if ( num == ROCK )  
        return "바위";  
    else  
        return "보";  
}
```

// 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

가비지 컬렉션

27

□ 가비지

- ▣ 가리키는 레퍼런스가 하나도 없는 객체
 - 누구도 사용할 수 없게 된 메모리

□ 가비지 컬렉션

- ▣ 자바 가상 기계에서 자동으로 가비지 수집하여 가용 메모리로 반환
- ▣ 가비지 컬렉션 스레드에 의해 수행

□ 개발자에 의한 강제 가비지 컬렉션

- ▣ System 또는 Runtime 객체의 gc() 메소드 호출

```
System.gc(); // 가비지 컬렉션 작동 요청
```

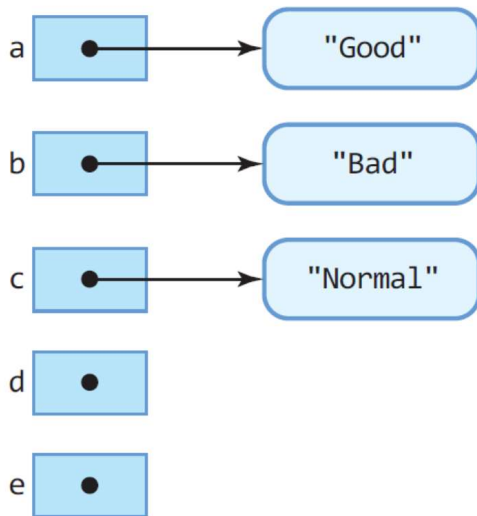
- 이 코드는 자바 가상 기계에 강력한 가비지 컬렉션 요청
 - 그러나 자바 가상 기계가 가비지 컬렉션 시점을 전적으로 판단

예제 4-9 : 가비지의 발생

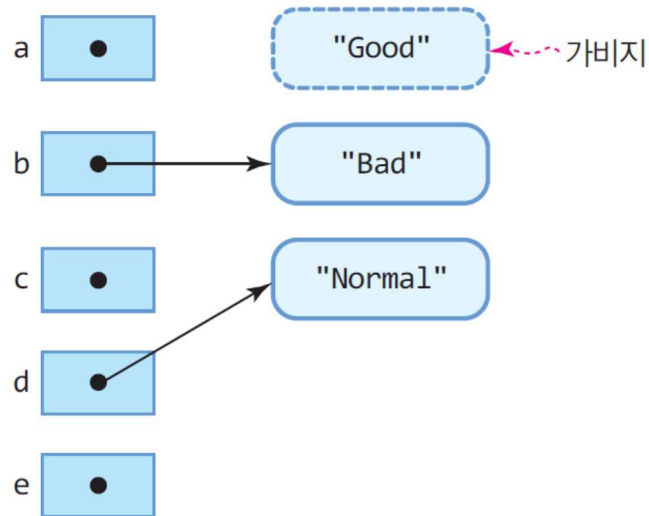
28

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)

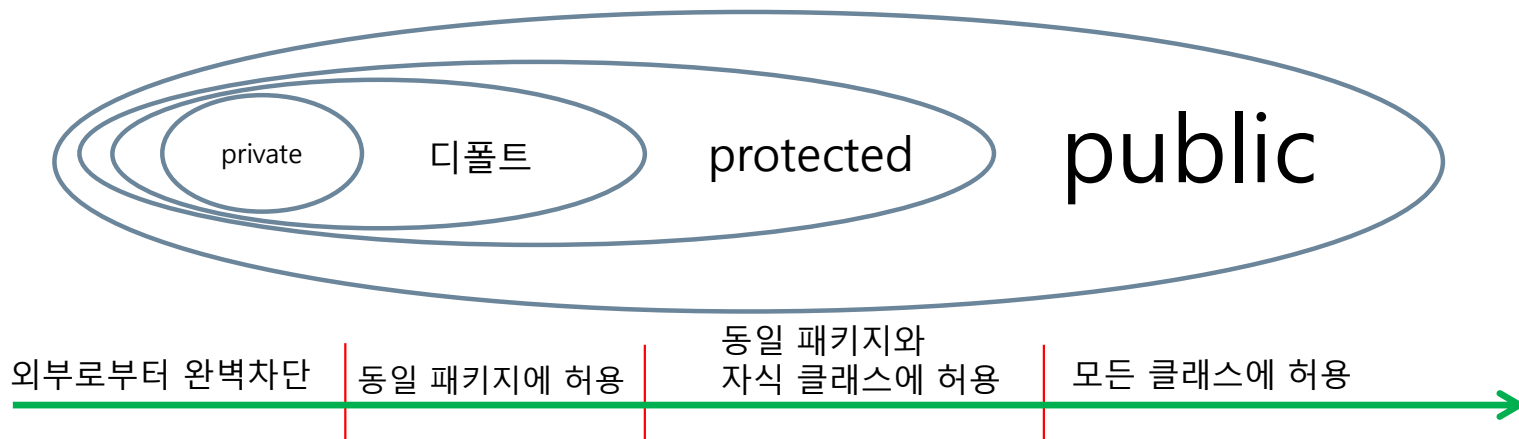


(b) 코드 전체 실행 후

접근 지정자

29

- 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



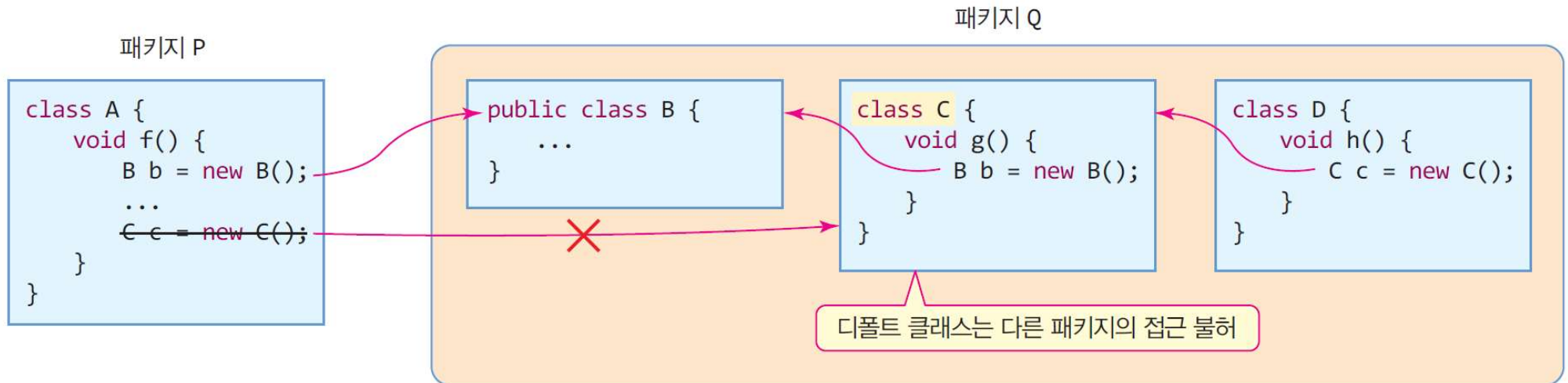
클래스 접근 지정

30

- 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

31

- ▣ 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 패키지와 자식 클래스	모든 클래스

멤버 접근 지정자의 이해

32

public 접근 지정 사례

패키지 P

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

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

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

private 접근 지정 사례

패키지 P

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

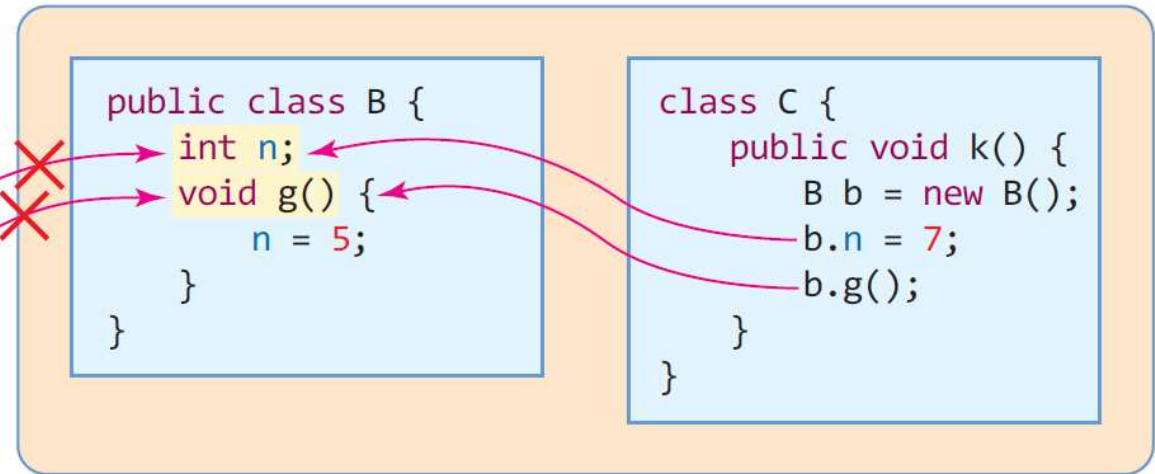
```
public class B {  
    private int n;  
    private void g() {  
        n = 5;  
    }  
}
```

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

디폴트 접근 지정 사례

패키지 P

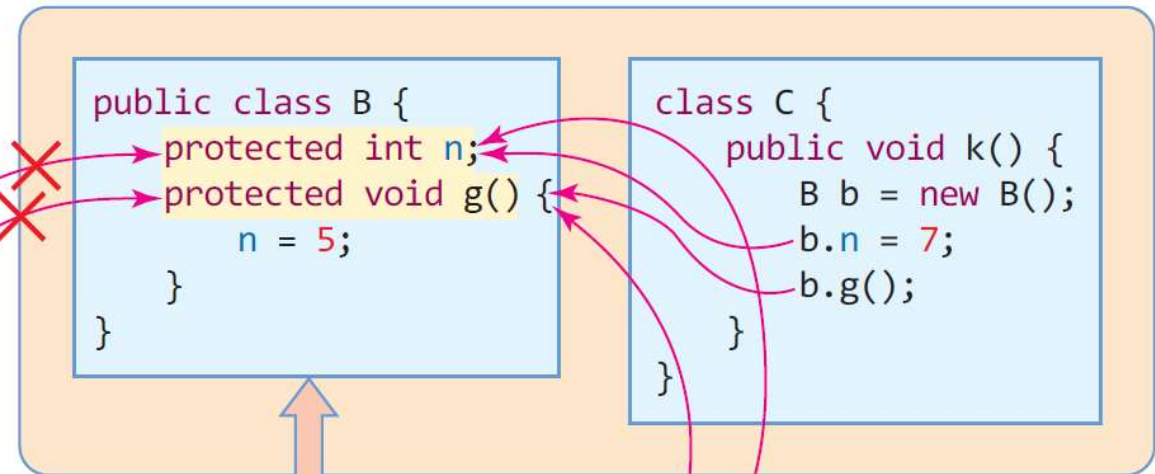
```
class A {
    void f() {
        B b = new B();
        b.n = 3;
        b.g();
    }
}
```



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();
    }
}
```


static 멤버와 non-static 멤버

34

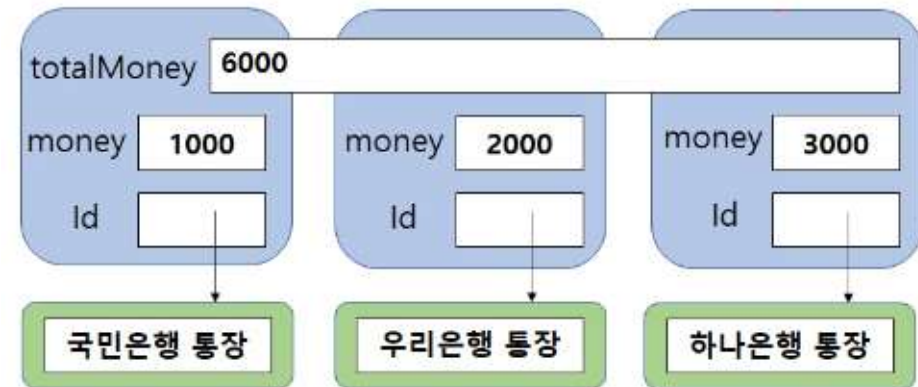
□ non-static 멤버의 특성

- 공간적 특성 - 멤버들은 객체마다 독립적으로 별도 존재
 - 인스턴스 멤버라고도 부름
- 시간적 특성 - 필드와 메소드는 객체 생성 후 비로소 사용 가능
- 비공유 특성 - 멤버들은 다른 객체에 의해 공유되지 않고 배타적

□ static 멤버란?

- 객체마다 생기는 것이 아님
- 클래스당 하나만 생성됨
 - 클래스 멤버라고도 부름
- 객체를 생성하지 않고 사용가능
- 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



non-static 멤버와 static 멤버의 차이

35

	non-static 멤버	static 멤버
선언	<pre>class Sample { int n; void g() {...} }</pre>	<pre>class Sample { static int m; static void f() {...} }</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

예제 static 멤버를 가진 통장 클래스

36

국민은행 통장, 우리은행 통장, 하나은행 통장을 가진 사람의 예금 총액을 계산

```
class Bankbook{
    private String id;
    private int money;
    private static int totalMoney = 0;

    public Bankbook( String id ){
        this.id = id;
        this.money = 0;
    }

    public int deposit( int money ){
        this.money += money;
        this.totalMoney += money;
        return this.money;
    }

    public static int getTotalMoney() {
        return totalMoney;
    }

    public String toString() {
        return this.id + " : " + this.money + "원\n총 예금액 : " + totalMoney + "원";
    }
}
```

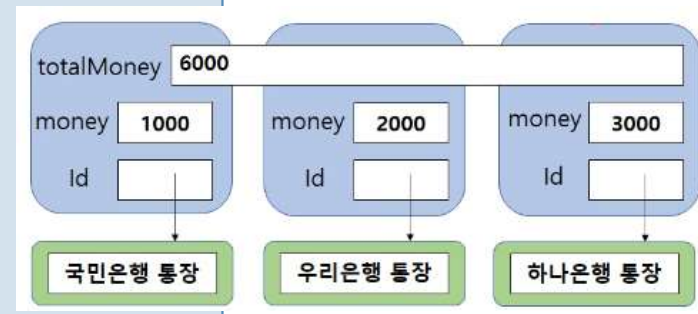
```
public class Bankbook_Main{
    public static void main(String[] args){
        Bankbook bankbook;

        bankbook = new Bankbook ( "국민은행 통장" );
        bankbook.deposit( 1000 );
        System.out.println( bankbook );

        bankbook = new Bankbook ( "우리은행 통장" );
        bankbook.deposit( 2000 );
        System.out.println( bankbook );

        bankbook = new Bankbook ( "하나은행 통장" );
        bankbook.deposit( 3000 );
        System.out.println( bankbook );

        System.out.println( "(전체 총 예금액) "
            + Bankbook.getTotalMoney() + "원" );
    }
}
```



static 메소드의 제약 조건 1

37

- ▣ 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
```

오류

```
    static int m;  
    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

38

- ▣ 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

final 필드

39

- final 필드, 상수 선언
 - ▣ 상수를 선언할 때 사용

```
class SharedClass {  
    public static final double PI = 3.14;  
}
```

- ▣ 상수 필드는 선언 시에 초기 값을 지정하여야 한다
- ▣ 상수 필드는 실행 중에 값을 변경할 수 없다

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

오류

final 클래스와 메소드

40

□ final 클래스 - 클래스 상속 불가

```
final class FinalClass {  
    ....  
}  
class SubClass extends FinalClass { // 컴파일 오류. FinalClass 상속 불가  
    ....  
}
```

□ final 메소드 - 오버라이딩 불가

```
public class SuperClass {  
    protected final int finalMethod() { ... }  
}  
  
class SubClass extends SuperClass { // SubClass가 SuperClass 상속  
    protected int finalMethod() { ... } // 컴파일 오류, 오버라이딩 할 수 없음  
}
```