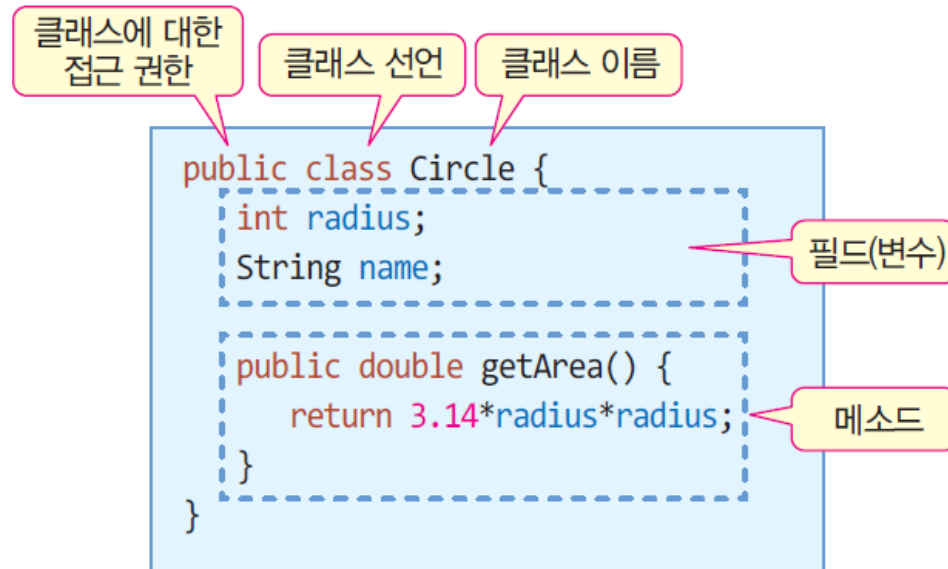


자바 클래스 구성

14

□ 클래스

- class 키워드로 선언
- 멤버 : 클래스 구성 요소
 - 필드(멤버 변수)와 메소드(멤버 함수)
- 클래스에 대한 public 접근 지정 : 다른 모든 클래스에서 클래스 사용 허락
- 멤버에 대한 public 접근 지정 : 다른 모든 클래스에게 멤버 접근 허용

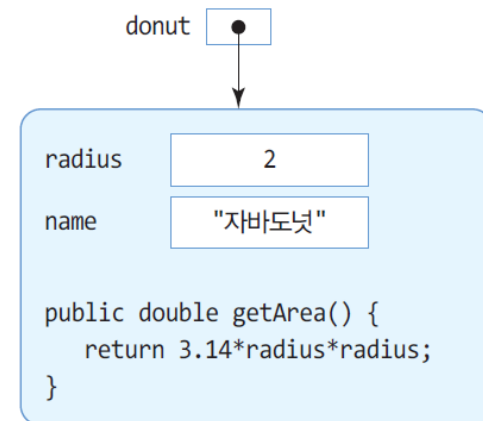
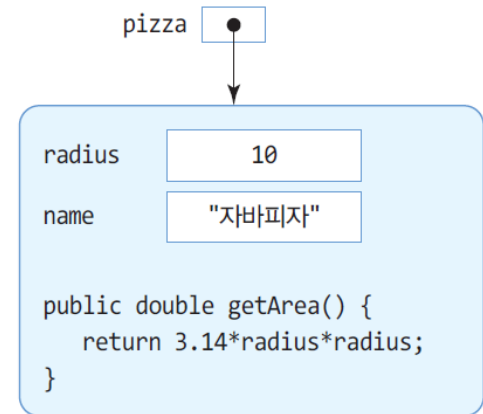


예제 4-1 : Circle 클래스의 객체 생성 및 활용

15

반지름과 이름을 가진 Circle 클래스를 작성하고, Circle 클래스의 객체를 생성하라.

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



자바피자의 면적은 314.0
자바도넛의 면적은 12.56

객체 생성과 활용

1. 레퍼런스 변수 선언
`Circle pizza;`

2. 객체 생성

- new 연산자 이용

`pizza = new Circle();`


3. 객체 멤버 접근

- 점(.) 연산자 이용

`pizza.radius = 10;`


`area = pizza.getArea();`

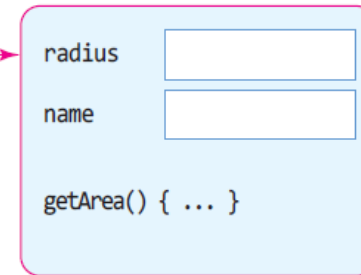
(1) `Circle pizza;`

pizza 

Circle 타입의 객체


(2) `pizza = new Circle();`

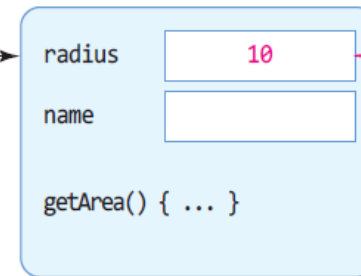
pizza 



객체 메모리
할당 및
객체 생성

(3) `pizza.radius = 10;`

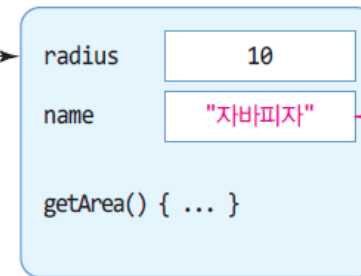
pizza 



radius 값 변경


(4) `pizza.name = "자바피자";`

pizza 

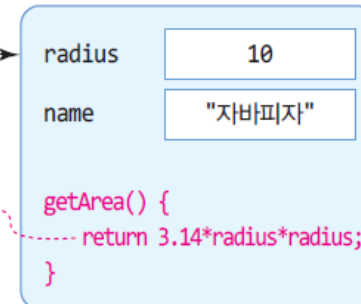


name 값 변경

(5) `double area = pizza.getArea();`

pizza 

area  314.0



getArea()
메소드 실행

예제 4-2 : Rectangle 클래스 만들기 연습

17

너비(width)와 높이(height) 필드, 그리고 면적 값을 제공하는 getArea() 메소드를 가진 Rectangle 클래스를 작성하라.

```
import java.util.Scanner;

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

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

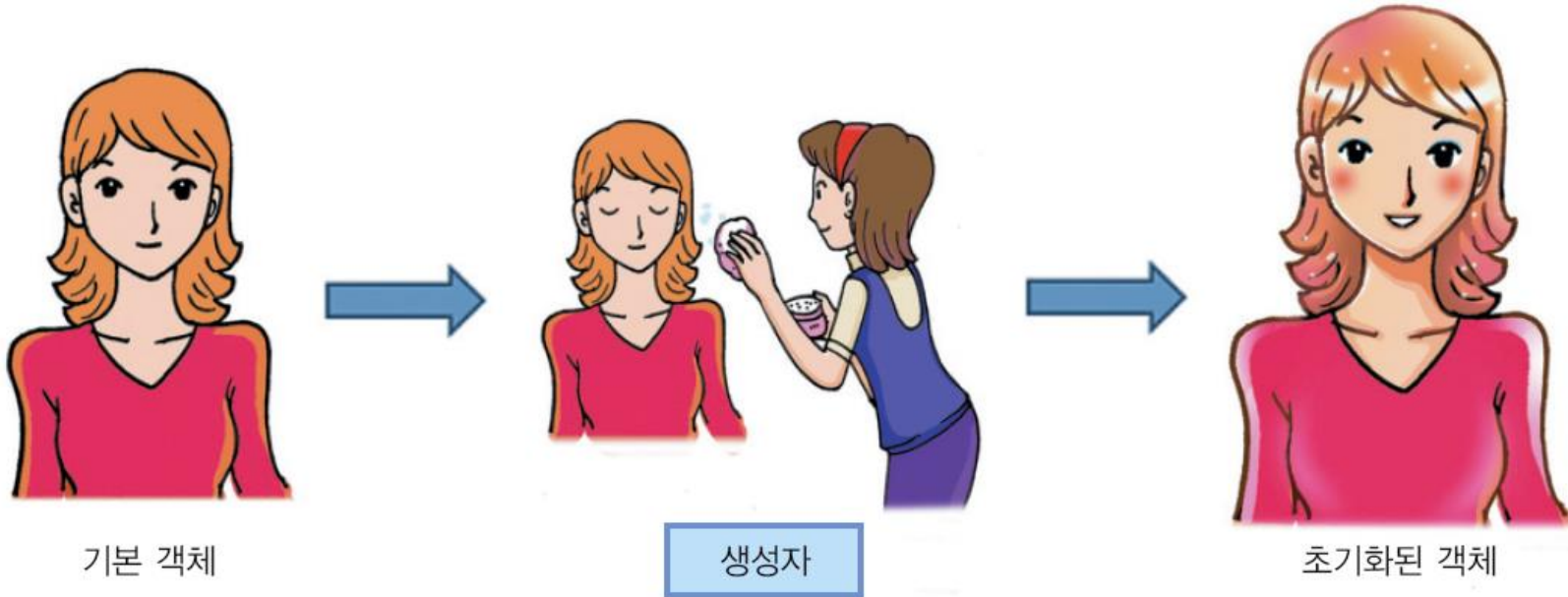
```
>> 4 5
사각형의 면적은 20
```

생성자 개념과 목적

18

□ 생성자

- ▣ 객체가 생성될 때 초기화 목적으로 실행되는 메소드
- ▣ 객체가 생성되는 순간에 자동 호출



예제 4-3 : 두 개의 생성자를 가진 Circle 클래스

19

두 개의 생성자를 가진 Circle 클래스를 만들어 보라.

```
public 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;
    }

    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);
    }
}
```

생성자 이름은 클래스 이름과 동일

생성자는 리턴 타입 없음

자바피자의 면적은 314.0
도넛피자의 면적은 3.14

생성자의 특징

20

- 생성자 이름은 클래스 이름과 동일
- 생성자는 여러 개 작성 가능(생성자 중복)

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

- 생성자는 객체 생성시 한 번만 호출
 - 자바에서 객체 생성은 반드시 new 연산자로 함

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

- 생성자의 목적은 객체 생성 시 초기화
- 생성자는 리턴 타입을 지정할 수 없음



```
public void Circle() {...} // 오류. void도 사용 안 됨
```

예제 4-4 : 생성자 선언 및 호출 연습

21

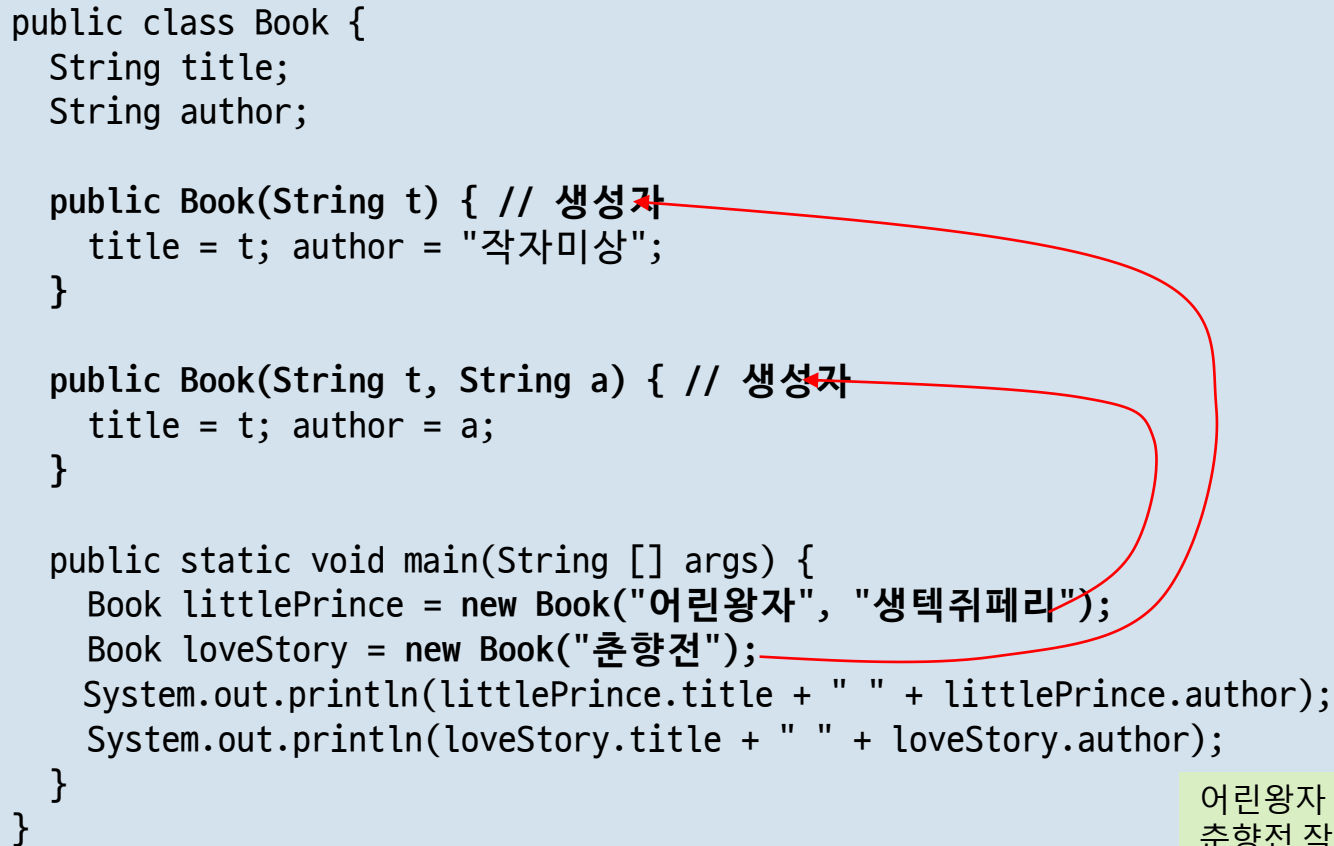
제목과 저자를 나타내는 title과 author 필드를 가진 Book 클래스를 작성하고, 생성자를 작성하여 필드를 초기화하라.

```
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

22

- 기본 생성자(default constructor)
 - ▣ 매개 변수 없고, 아무 작업 없이 단순 리턴하는 생성자

```
class Circle {  
    public Circle() { } // 기본 생성자  
}
```

- ▣ 디폴트 생성자라고도 불림

기본 생성자가 자동 생성되는 경우

23

- 클래스에 생성자가 하나도 선언되어 있지 않을 때
 - ▣ 컴파일러에 의해 기본 생성자 자동 생성

```
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(5);  
        System.out.println(pizza.getArea());  
    }  
}
```

컴파일러에 의해 기본
생성자 자동 삽입

public Circle() { }

호출

기본 생성자가 자동 생성되지 않는 경우

24

- 클래스에 생성자가 선언되어 있는 경우
 - ▣ 컴파일러는 기본 생성자를 자동 생성해 주지 않는다.

```
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;  
    }
```

호출

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

오류

```
    Circle donut = new Circle();  
    System.out.println(donut.getArea());  
}
```

오류 메시지 : The constructor
Circle() is undefined

this 레퍼런스

25

□ this

▣ 객체 자신에 대한 레퍼런스

- 컴파일러에 의해 자동 관리, 개발자는 사용하기만 하면 됨
- **this.멤버** 형태로 멤버를 접근할 때 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

객체 속에서의 this

26

```
public class Circle {  
    int radius;  
    public Circle(int radius) {  
        this.radius = radius;  
    }  
    void set(int radius) {  
        this.radius = radius;  
    }  
  
    public static void main(String[] args) {  
        Circle ob1 = new Circle(1);  
        Circle ob2 = new Circle(2);  
        Circle ob3 = new Circle(3);  
  
        ob1.set(4);  
        ob2.set(5);  
        ob3.set(6);  
    }  
}
```

ob1



radius ~~1~~ 4
...
void set(int radius) {
 this.radius = radius;
}

ob2



radius ~~2~~ 5
...
void set(int radius) {
 this.radius = radius;
}

ob3



radius ~~3~~ 6
...
void set(int radius) {
 this.radius = radius;
}

this 요약

27



this()로 다른 생성자 호출

28

□ this()

- 같은 클래스의 다른 생성자 호출
- 생성자 내에서만 사용 가능
- 생성자 코드의 제일 처음에 있어야 함
 - this() 사용 실패 사례



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

예제 4-5 this()로 다른 생성자 호출

29

예제 4-4에서 작성한 Book 클래스의 생성자를 this()를 이용하여 수정하라.

```
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

객체 배열

30

- 자바의 객체 배열
 - ▣ 객체에 대한 레퍼런스 배열임
- 자바의 객체 배열 만들기 3 단계
 - ❶ 배열 레퍼런스 변수 선언
 - ❷ 레퍼런스 배열 생성
 - ❸ 배열의 각 원소 객체 생성

```
Circle [] c; ❶ Circle 배열에 대한 레퍼런스 변수 c 선언
c = new Circle[5]; ❷ 레퍼런스 배열 생성

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

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

배열의 원소 객체 사용

객체 배열 선언과 생성 과정

31

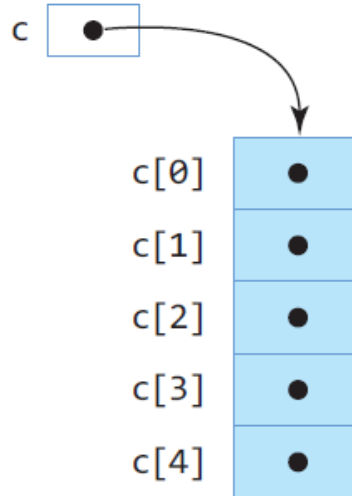
❶ 배열에 대한 레퍼런스 변수 선언

```
Circle[] c;
```



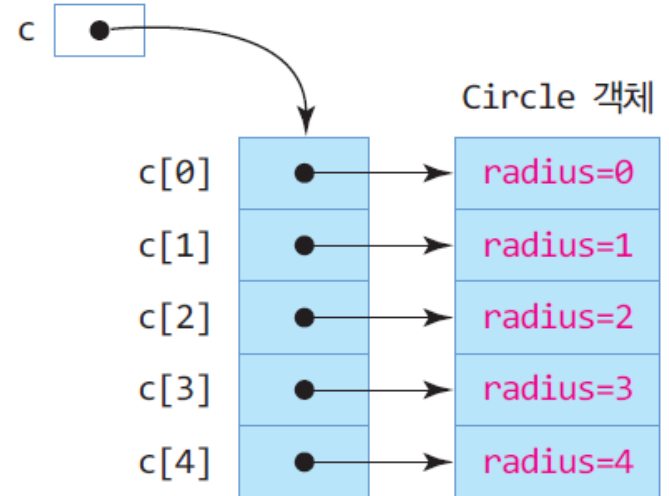
❷ 레퍼런스 배열 생성

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



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

❸ 객체 생성



예제 4-6 : Circle 배열 만들기

32

Circle 객체 5개를 가지는 배열을 생성하고, Circle 객체의 반지름을 0에서 4까지 각각 지정한 후, 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()) + " ");
    }
}
```

0 3 12 28 50

예제 4-7 : 객체 배열 만들기 연습

33

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();
    }
}
```

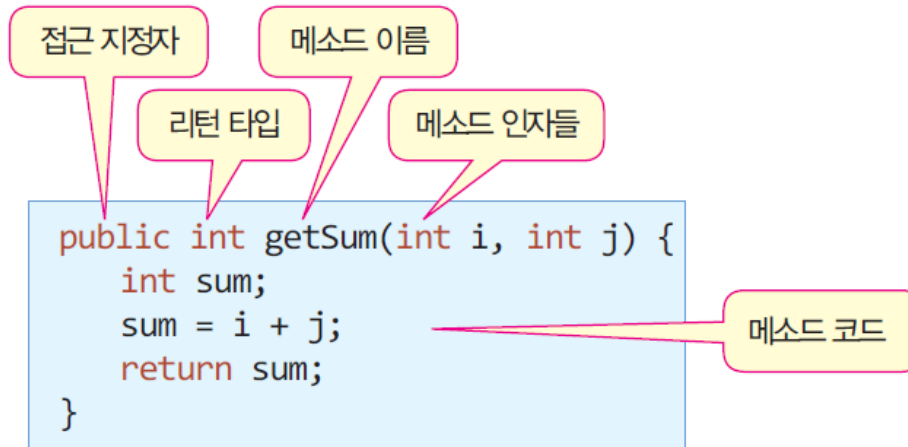
메소드

34

□ 메소드

- ▣ 메소드는 C/C++의 함수와 동일
- ▣ 자바의 모든 메소드는 반드시 클래스 안에 있어야 함(캡슐화 원칙)

□ 메소드 형식



▣ 접근 지정자

- 다른 클래스에서 메소드를 접근할 수 있는지 여부 선언
- `public`, `private`, `protected`, 디폴트(접근 지정자 생략)

▣ 리턴 타입

- 메소드가 리턴하는 값의 데이터 타입

인자 전달 – 기본 타입의 값이 전달되는 경우

- 매개 변수가 byte, int, double 등 기본 타입으로 선언되었을 때
 - 호출자가 건네는 값이 매개 변수에 복사되어 전달. 실인자 값은 변경되지 않음

⇒ 실행 결과

10

```
public class CallByValue {  
    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

```
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++;  
}
```

→ 실행 결과

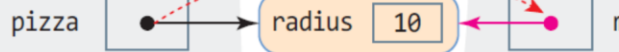
11

main() 실행 시작

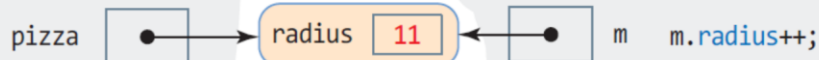
pizza = new Circle(10);



increase(pizza);



increase(int m) 실행 시작



increase(int m) 종료

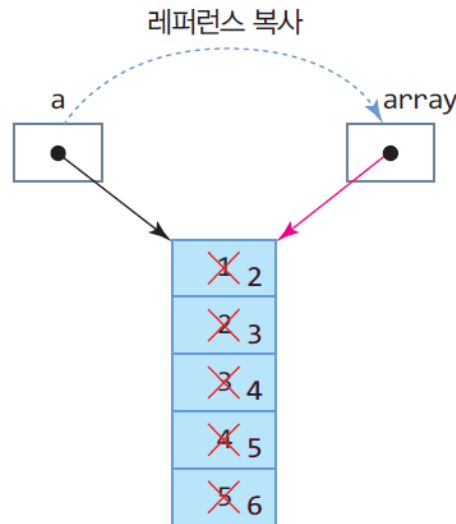
System.out.println(pizza.radius);



인자 전달 - 배열이 전달되는 경우

- 배열 레퍼런스만 매개 변수에 전달
 - 배열 통째로 전달되지 않음
- 객체가 전달되는 경우와 동일
 - 매개 변수가 실인자의 배열 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]+" ");  
    }  
}
```



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

⇒ 실행 결과

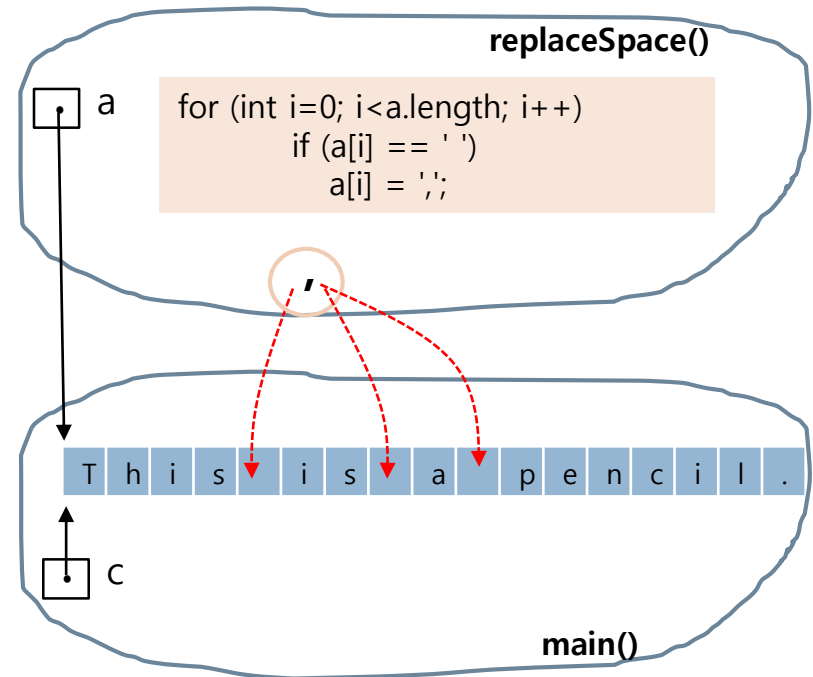
2 3 4 5 6

예제 4-8 : 인자로 배열이 전달되는 예

38

char[] 배열을 전달받아 배열 속의 공백(' ') 문자를 ';'로 대체하는 메소드를 작성하라.

```
public class ArrayParameter {  
    static void replaceSpace(char a[]) {  
        for (int i=0; i<a.length; i++)  
            if (a[i] == ' ')  
                a[i] = ';'  
    }  
    static void printCharArray(char a[]) {  
        for (int i=0; i<a.length; i++)  
            System.out.print(a[i]);  
        System.out.println();  
    }  
    public static void main (String args[]) {  
        char c[] = {'T','h','i','s',' ','i','s',' ','a',' ','p','e','n','c','i','l','.'};  
        printCharArray(c);  
        replaceSpace(c);  
        printCharArray(c);  
    }  
}
```



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

메소드 오버로딩

39

- 오버로딩(Overloading)
 - ▣ 한 클래스 내에서 두 개 이상의 이름이 같은 메소드 작성
 - 메소드 이름이 동일하여야 함
 - 매개 변수의 개수가 서로 다르거나, 타입이 서로 달라야 함
 - 리턴 타입은 오버로딩과 관련 없음

성공한 오버로딩과 메소드 호출

40

```
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;  
    }  
}
```

오버로딩 실패 사례

41

오류

```
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);
    }
}
```

매개 변수의 개수와 타입
이 같기 때문에 오버로딩
실패

객체 치환 시 주의할 점

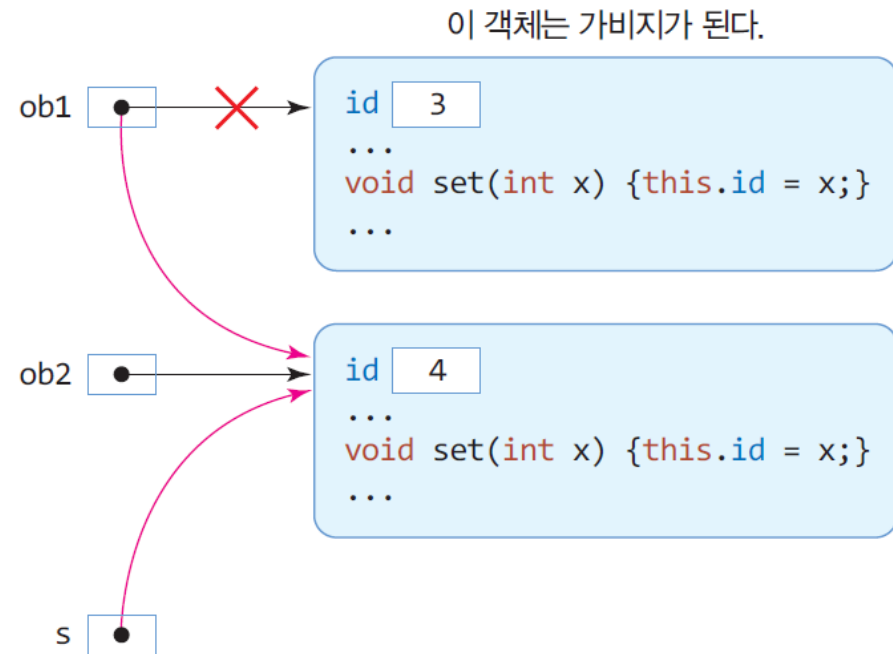
42

* 객체 치환은 객체 복사가 아니며, 레퍼런스의 복사이다.

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

객체 치환

ob1.id=4
ob2.id=4



객체 소멸

43

□ 객체 소멸

- ▣ new에 의해 할당 받은 객체와 배열 메모리를 자바 가상 기계로 되돌려 주는 행위
- ▣ 소멸된 객체 공간은 가용 메모리에 포함

□ 자바에서 사용자 임의로 객체 소멸안됨

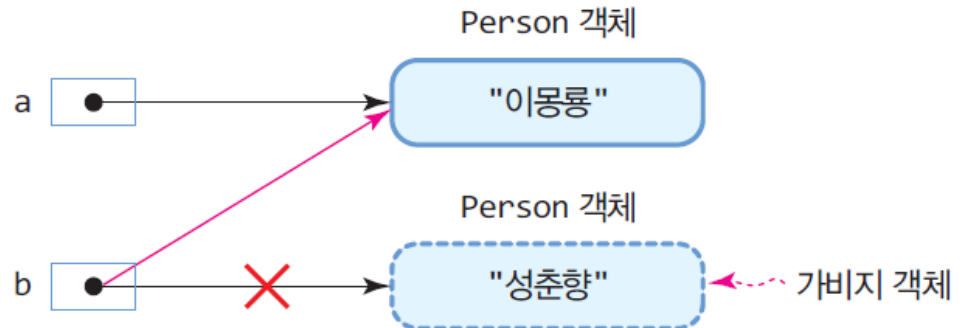
- ▣ 자바는 객체 소멸 연산자 없음
 - 객체 생성 연산자 : new
- ▣ 객체 소멸은 자바 가상 기계의 고유한 역할
- ▣ 자바 개발자에게는 매우 다행스러운 기능
 - C/C++에서는 할당 받은 객체를 개발자가 프로그램 내에서 삭제해야 함
 - C/C++의 프로그램 작성을 어렵게 만드는 요인
 - 자바에서는 사용하지 않는 객체나 배열을 돌려주는 코딩 책임으로부터 개발자 해방

가비지

44

- 가비지
 - ▣ 가리키는 레퍼런스가 하나도 없는 객체
 - 더 이상 접근할 수 없어 사용할 수 없게 된 메모리
- 가비지 컬렉션
 - ▣ 자바 가상 기계의 가비지 컬렉터가 자동으로 가비지 수집, 반환

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

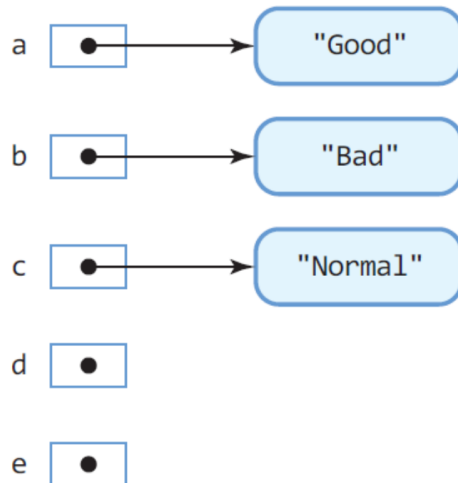


예제 4-9 : 가비지의 발생

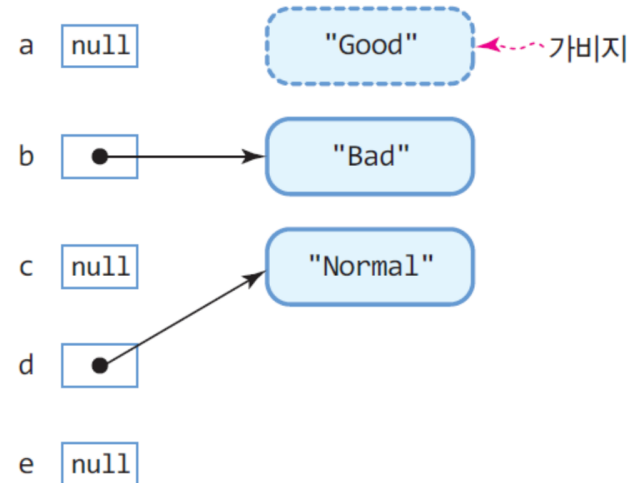
45

다음 소스에서 언제 가비지가 발생하는지 설명하라.

```
public class GarbageEx {  
    public static void main(String[] args) {  
        String a = new String("Good");  
        String b = new String("Bad");  
        String c = new String("Normal");  
        String d, e;  
        a = null;  
        d = c;  
        c = null;  
    }  
}
```



(a) 초기 객체 생성 시(라인 6까지)



(b) 코드 전체 실행 후

가비지 컬렉션

46

□ 가비지 컬렉션

- ▣ 자바 가상 기계가 가비지 자동 회수
 - 가용 메모리 공간이 일정 이하로 부족해질 때
 - 가비지를 수거하여 가용 메모리 공간으로 확보
- ▣ 가비지 컬렉터(garbage collector)에 의해 자동 수행

□ 강제 가비지 컬렉션 강제 수행

- ▣ 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

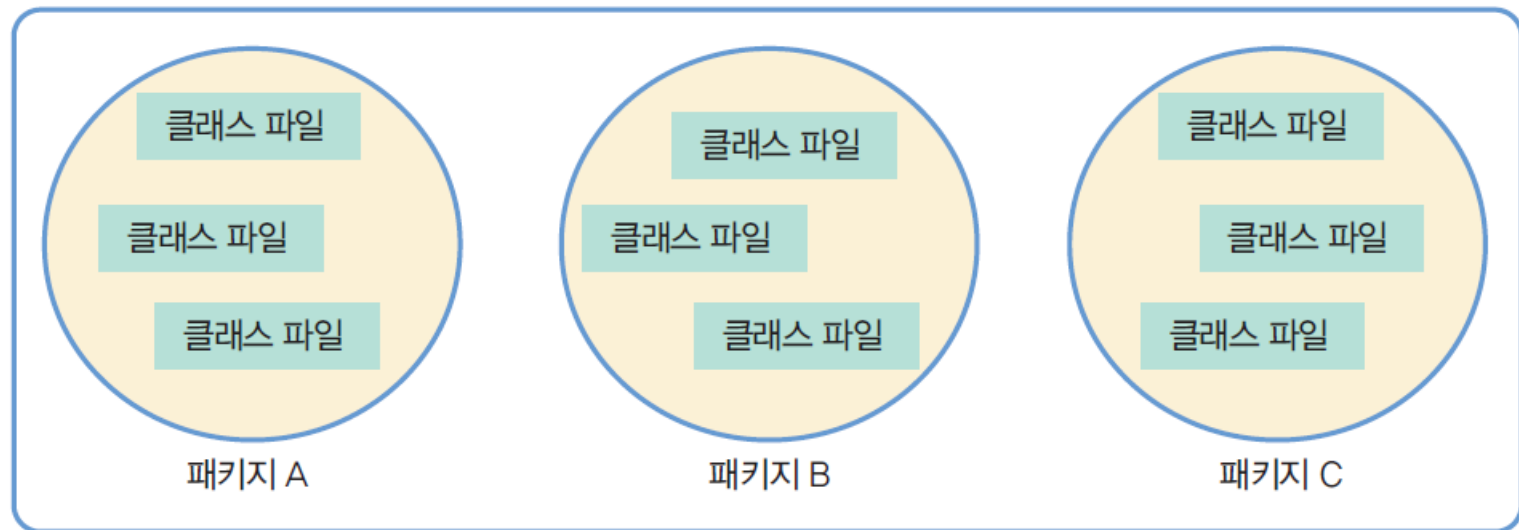
자바의 패키지 개념

47

□ 패키지

- 상호 관련 있는 클래스 파일(컴파일된 .class)을 저장하여 관리하는 디렉터리
- 자바 응용프로그램은 하나 이상의 패키지로 구성

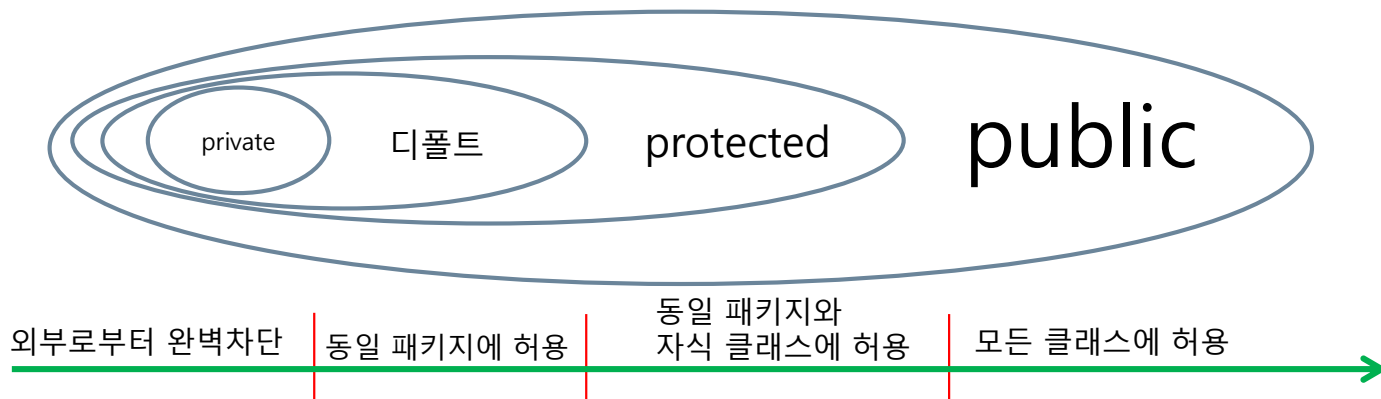
자바 응용프로그램



접근 지정자

48

- 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



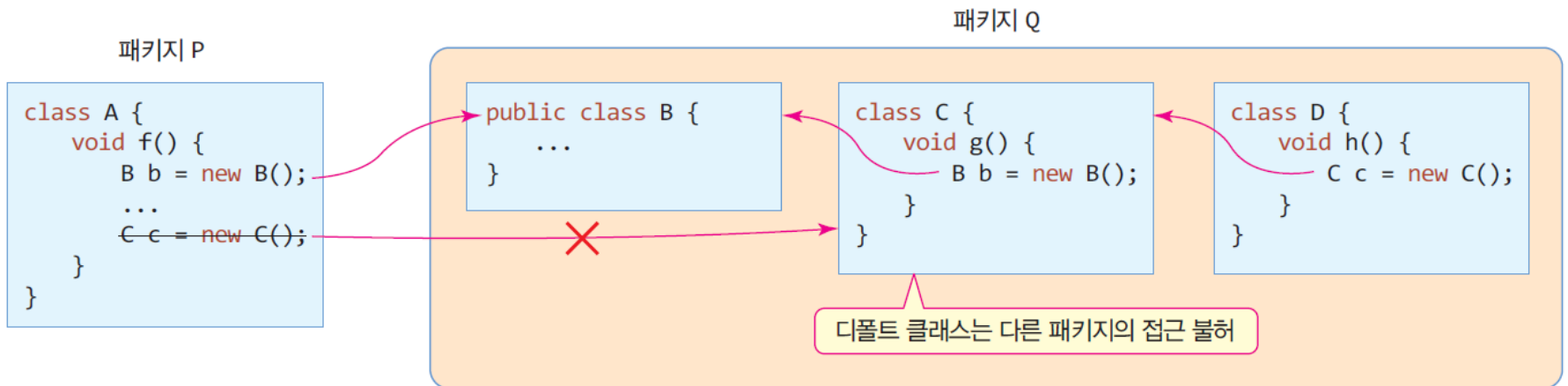
클래스 접근 지정

49

- 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

50

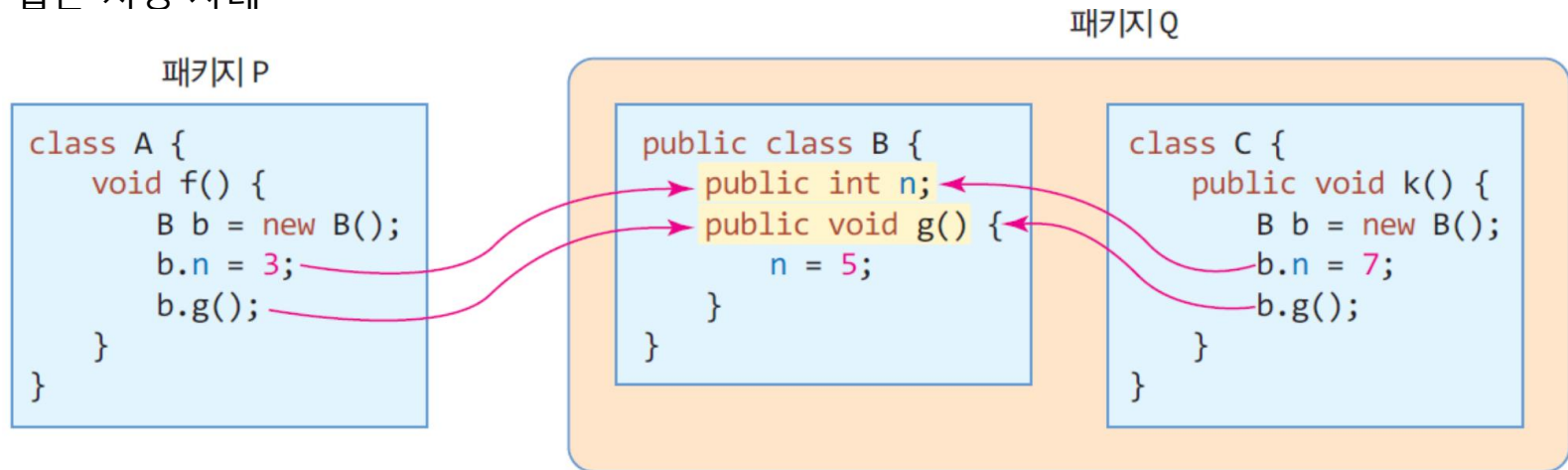
- ▣ public 멤버
 - 패키지에 관계 없이 모든 클래스에게 접근 허용
- ▣ private 멤버
 - 동일 클래스 내에만 접근 허용
 - 상속 받은 서브 클래스에서 접근 불가
- ▣ protected 멤버
 - 같은 패키지 내의 다른 모든 클래스에게 접근 허용
 - 상속 받은 서브 클래스는 다른 패키지에 있어도 접근 가능
- ▣ 디폴트(default) 멤버
 - 같은 패키지 내의 다른 클래스에게 접근 허용

| 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 |

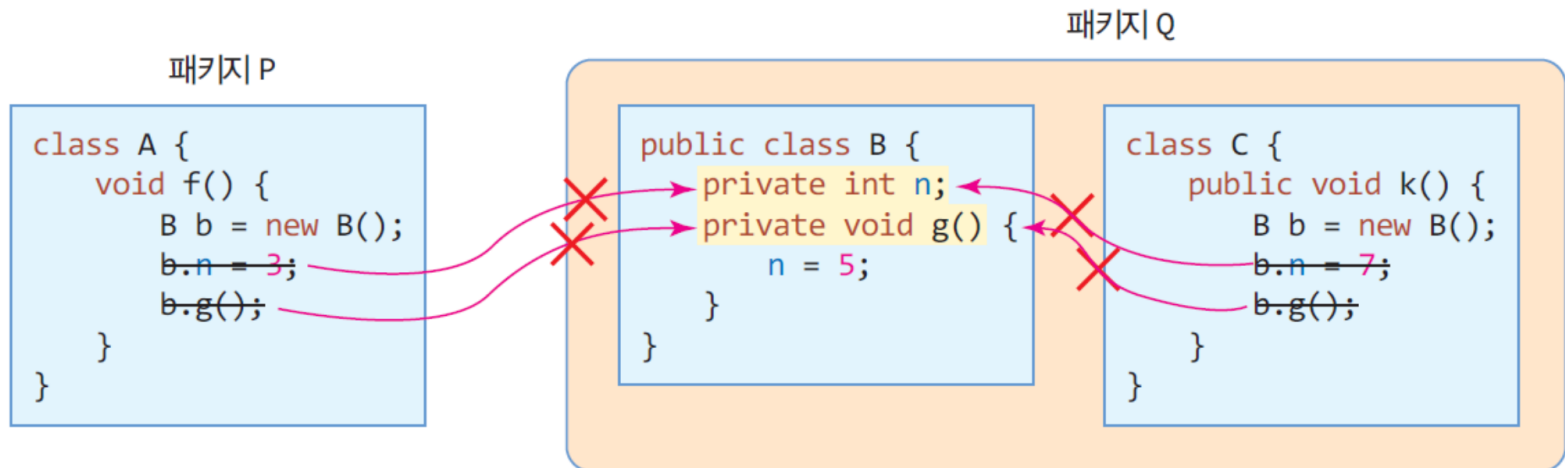
멤버 접근 지정 사례

51

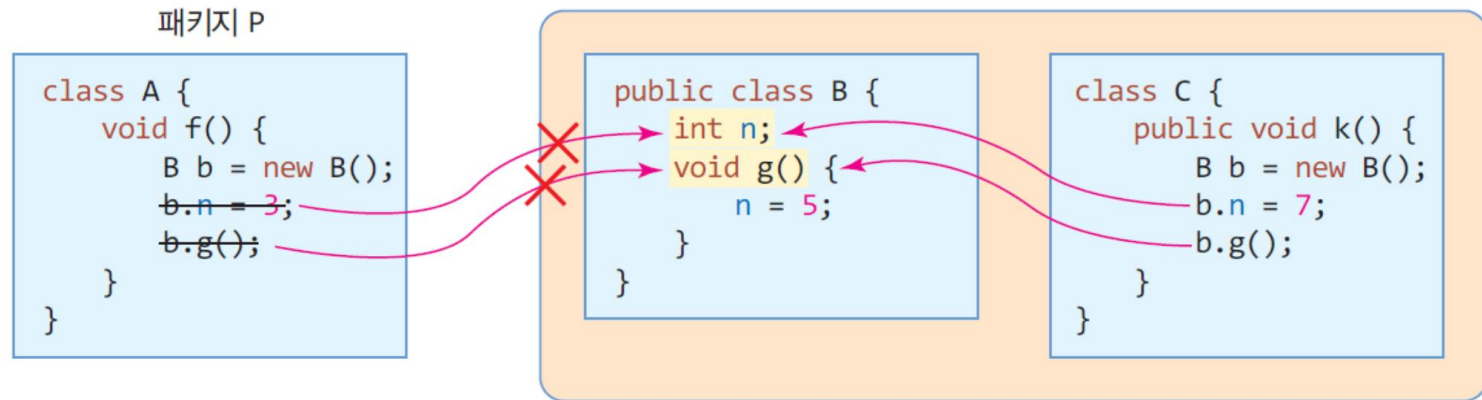
public 접근 지정 사례



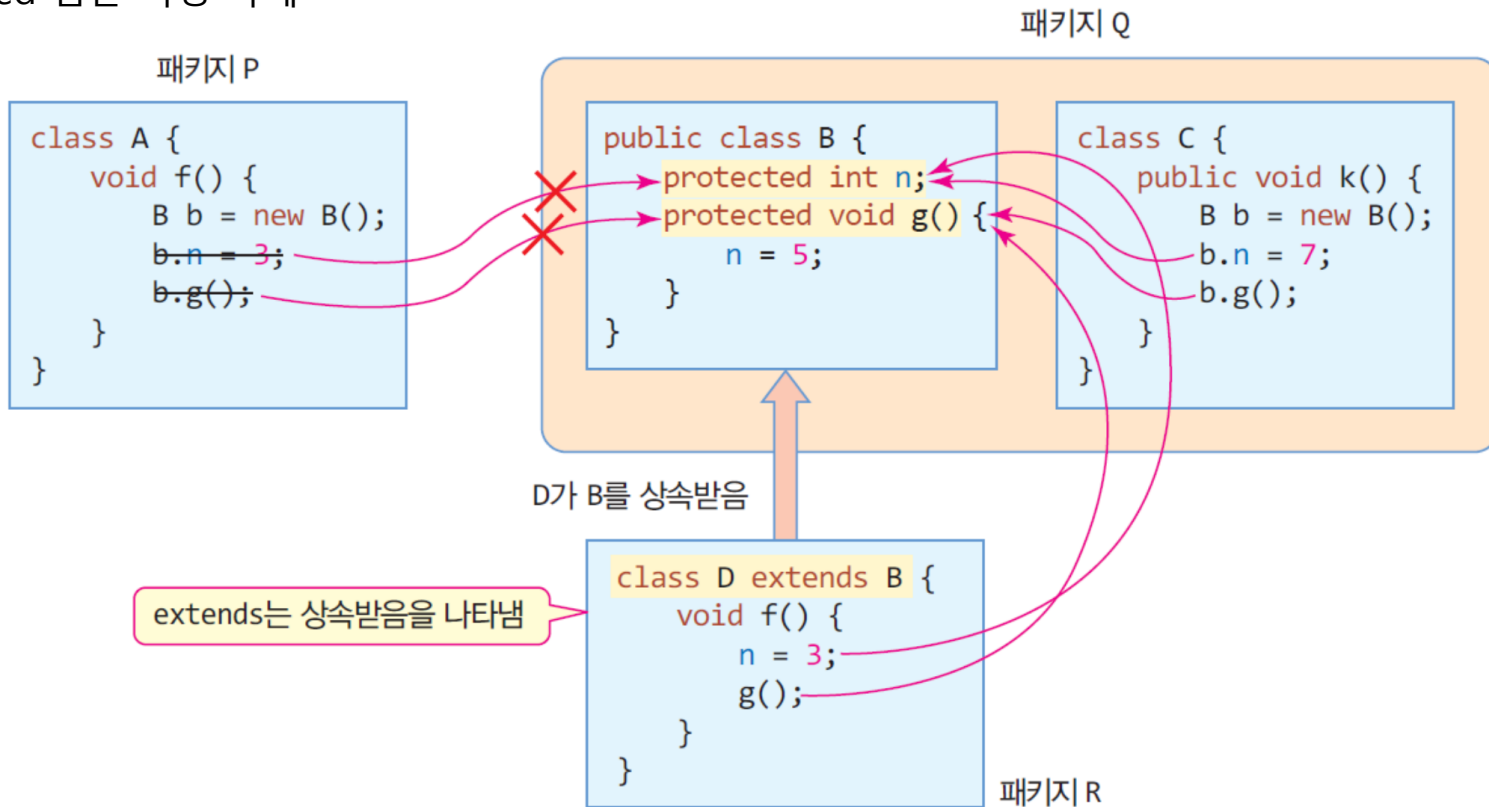
private 접근 지정 사례



디폴트 접근 지정 사례



protected 접근 지정 사례



예제 4-10 : 멤버의 접근 지정자

53

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)
```


static 멤버

54

□ static 멤버 선언

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

□ 객체 생성과 non-static 멤버의 생성

- non-static 멤버는 객체가 생성될 때, 객체마다 생긴다

```
class A {  
    int n;  
    void g() {...}  
}
```

```
A a1 = new A();  
A a2 = new A();  
A a3 = new A();
```

n
g()
a1

n
g()
a2

n
g()
a3

객체마다 n, g()의 non-static 멤버들이 생긴다

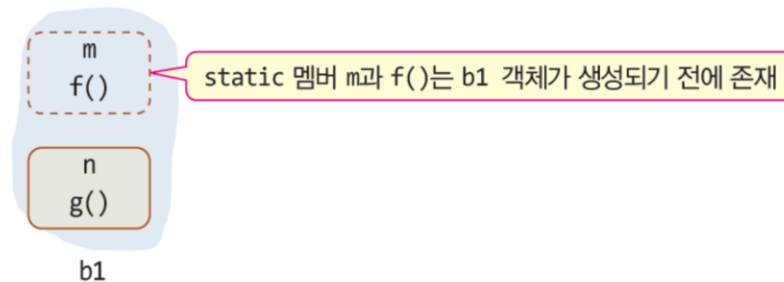
static 멤버의 생성

55

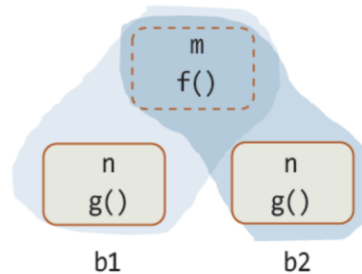
- static 멤버는 클래스당 하나만 생성
- 객체들에 의해 공유됨

```
class StaticSample {  
    int n;  
    void g() {...}  
    static int m;  
    static void f() {...}  
}
```

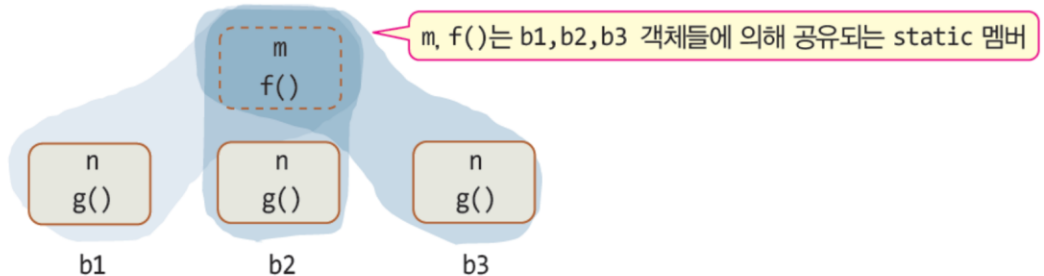
StaticSample b1 = new StaticSample(); 생성 후



StaticSample b2 = new StaticSample(); 생성 후



StaticSample b3 = new StaticSample(); 생성 후



static 멤버와 non-static 멤버 특성 정리

56

| | 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 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 멤버 사용

57

□ 클래스 이름으로 접근 가능

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

□ 객체의 멤버로 접근 가능

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

□ non-static 멤버는 클래스 이름으로 접근 안 됨



```
StaticSample.n = 5; // n은 non-static이므로 컴파일 오류  
StaticSample.g();   // g()는 non-static이므로 컴파일 오류
```

static의 활용

58

1. 전역 변수와 전역 함수를 만들 때 활용
2. 공유 멤버를 만들고자 할 때
 - ▣ static으로 선언한 멤버는 클래스의 객체들 사이에 공유

예제 4-11 : static 멤버를 가진 Calc 클래스 작성

59

전역 함수로 작성하고자 하는 abs, max, min의 3개 함수를 static 메소드를 작성하고 호출하는 사례를 보여라.

```
class Calc {  
    public static int abs(int a) { return a>0?a:-a; }  
    public static int max(int a, int b) { return (a>b)?a:b; }  
    public static int min(int a, int b) { return (a>b)?b:a; }  
}  
  
public class CalcEx {  
    public static void main(String[] args) {  
        System.out.println(Calc.abs(-5));  
        System.out.println(Calc.max(10, 8));  
        System.out.println(Calc.min(-3, -8));  
    }  
}
```

5
10
-8

static 메소드의 제약 조건 1

60


- ▣ static 메소드는 오직 static 멤버만 접근 가능
 - 객체가 생성되지 않은 상황에서도 static 메소드는 실행될 수 있기 때문에, non-static 멤버 활용 불가
 - 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;  
    static void s1(int x) {n = x;} // 컴파일 오류. static 메소드는 non-static 필드 n 사용 불가  
    static void s2(int x) {f1(3);} // 컴파일 오류. static 메소드는 non-static 메소드 f1() 사용 불가  
    static void s3(int x) {m = x;} // 정상. static 메소드는 static 필드 m 사용 가능  
    static void s4(int x) {s3(3);} // 정상. static 메소드는 static 메소드 s3() 호출 가능  
}
```

static 메소드의 제약 조건 2

61

- ▣ static 메소드는 this 사용불가
 - static 메소드는 객체 없이도 사용 가능하므로, this 레퍼런스 사용할 수 없음


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

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


final 클래스와 메소드

62

□ final 클래스 - 더 이상 클래스 상속 불가능

```
final class FinalClass {  
    ....  
}  
 class DerivedClass extends FinalClass { // 컴파일 오류  
    ....  
}
```

□ final 메소드 - 더 이상 오버라이딩 불가능

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

final 필드

63

- 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