객체지향 프로그래밍

Object-Oriented Programming

본 자료는 한빛아카데미에서 제공한 강의자료를 기반으로 재구성하였습니다.

동국대학교 dongguk university

Chapter 06.
상속

상속

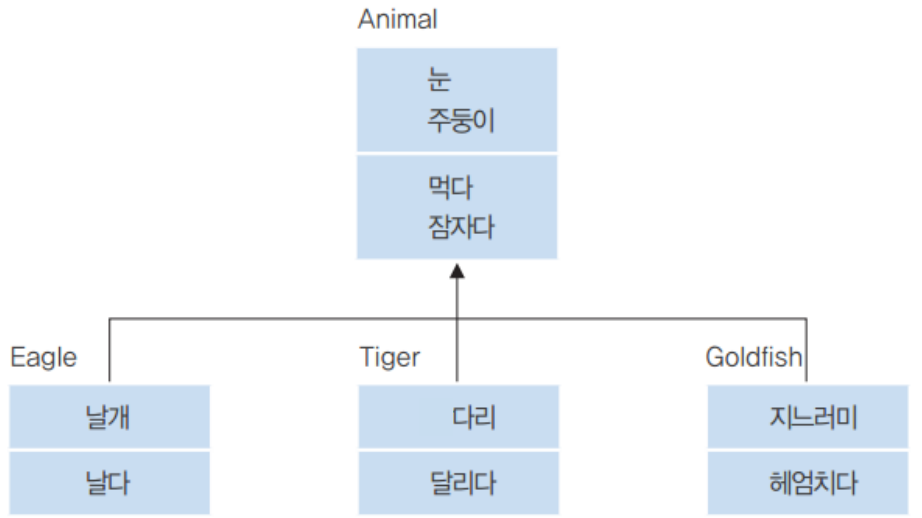
상속의 필요성

- 공통된 특성을 띄는 서로 다른 클래스가 존재
  (Eagle, Tiger, Goldfish)

- 상속이 없다면 클래스마다 중복된 내용 작성

- 상속을 도입해 공통된 부분을 통합
  → 중복 코드 제거, 유지 보수 용이

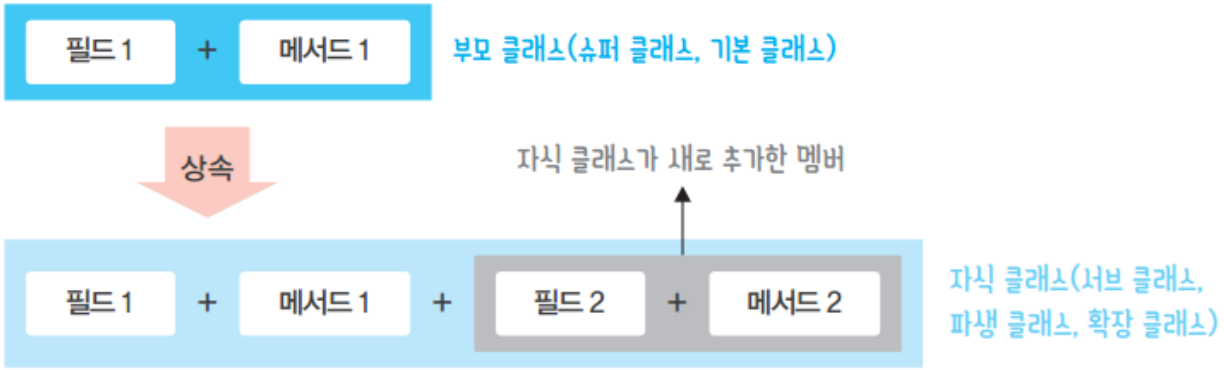
(a) 상속을 적용하기 전



(b) 상속을 적용한 후

상속

- 자식 클래스는 부모 클래스에서 물려받은 멤버를 그대로 사용하거나 변경할 수 있고, 새로운 멤버도 추가할 수 있다.
- 따라서 자식 클래스는 대체로 부모 클래스보다 속성이나 동작이 더 많다.



상속

부모 · 자식 클래스의 관계

- 상속은 is-a 관계

is-a(상속 관계)	has-a(소유 관계)
• 원은 도형이다. • 사과는 과일이다. • Tandem은 Bike다.	• 자동차는 엔진이 있다. • 스마트폰은 카메라가 있다. • 컴퓨터는 마우스가 있다.

상속

상속의 선언

- **extends** 키워드 사용

부모 클래스

부모 클래스 이름

```
class SuperClass {
    // 필드
    // 메서드
}
```

상속

자식 클래스

자식 클래스 이름 / 부모 클래스 이름

```
class SubClass extends SuperClass {
    // 필드
    // 메서드
}
```

※ **자바에서는 다중 상속 지원하지 않음.**

상속

다중 상속

- Java에서 다중 상속은 금지되어 있음

```
class SubClass extends SuperClass1, SuperClass2 {
}
```

상속

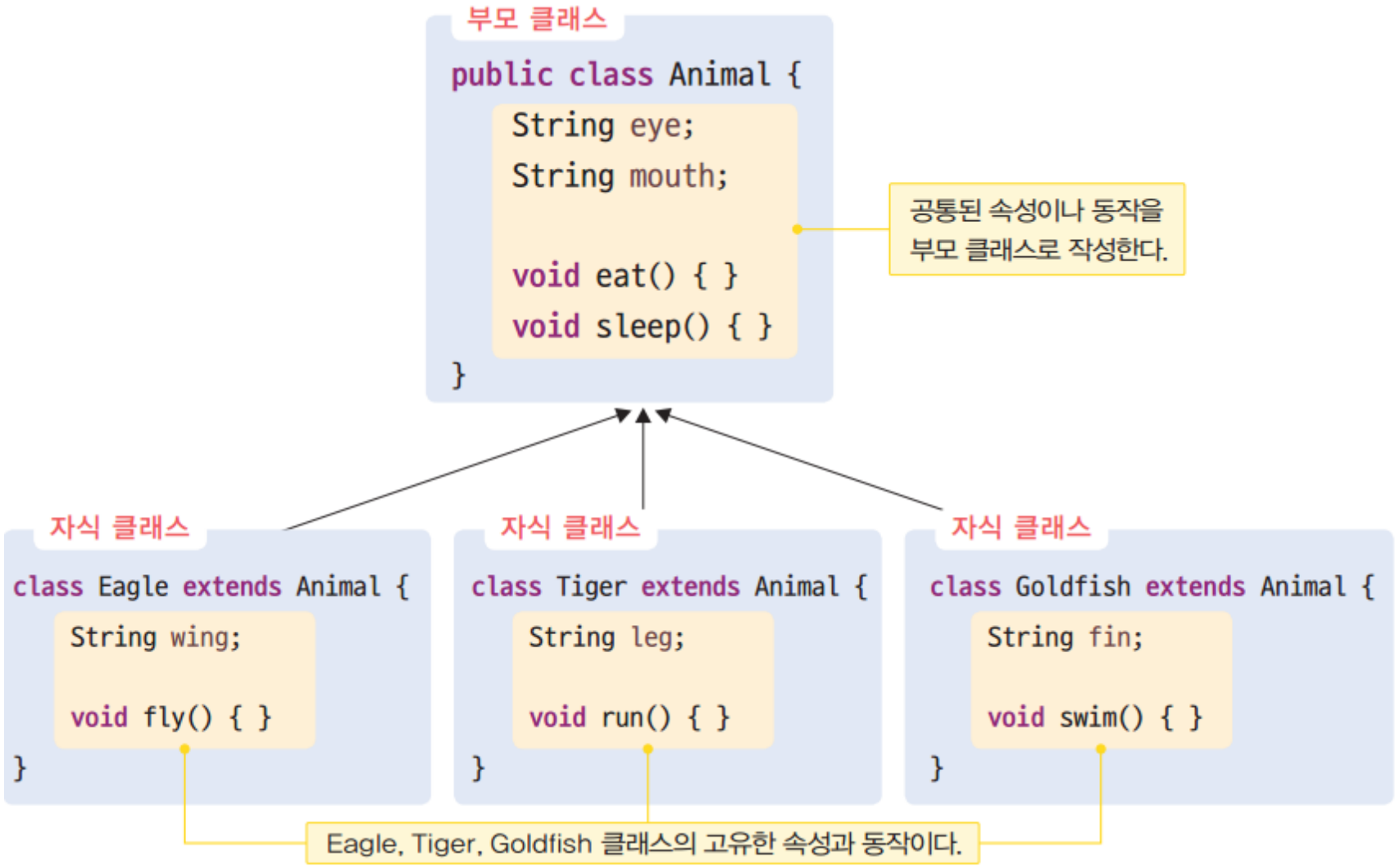
현실 세계와 상속 적용

- 객체 지향의 상속을 적용할 수 있는 현실 세계의 예

부모 클래스	자식 클래스
Animal	Eagle, Tiger, Goldfish
Bike	MountainBike, RoadBike, TandemBike
Circle	Ball, Cone, Cylinder
Drinks	Beer, Coke, Juice, Wine
Employee	RegularEmployee, TemporaryEmployee, ContractEmployee

상속

현실 세계와 상속 적용



상속

Practice

예제 6-1: Circle 클래스

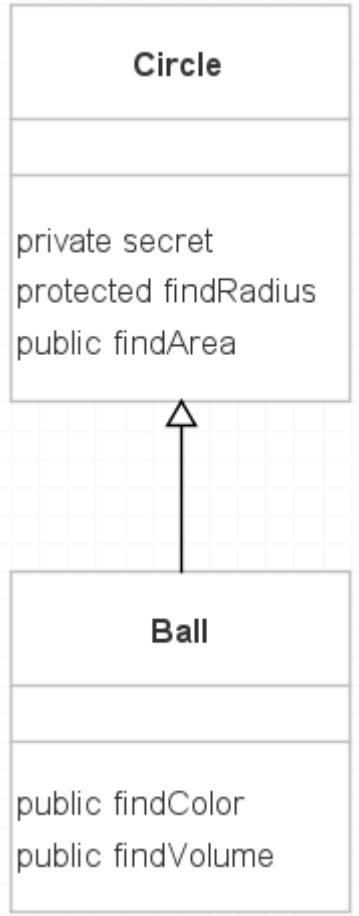
```java
package chap06;

public class Circle {

    // 클래스 내부에서만 접근 허용
    private void secret() {
        System.out.println("비밀이다.");
    }

    // 부모, 자식 클래스에게만 접근 허용
    protected void findRadius() {
        System.out.println("반지름이 10.0센티이다.");
    }

    public void findArea() {
        System.out.println("넓이는 (π*반지름*반지름)이다.");
    }
}
```

상속

Practice

예제 6-2: Circle 클래스의 자식 Ball 클래스

```java
package chap06;

public class Ball extends Circle {
    private String color;

    public Ball(String color) {
        this.color = color;
    }

    public void findColor( ) {
        System.out.println(color + " 공이다.");
    }

    public void findVolume( ) {
        System.out.println("부피는 4/3*(π*반지름*반지름*반지름)이다.");
    }
}
```

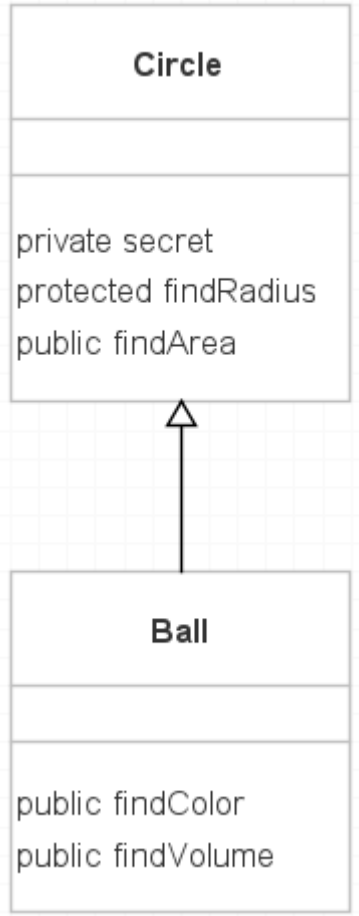


상속

Practice

예제 6-2: Circle 클래스의 자식 Ball 클래스

```java
package chap06;

public class Ball extends Circle {
    private String color;          //Ball 클래스에 추가한 필드이다.

    public Ball(String color) {
        this.color = color;
    }                              //Ball 클래스에 추가한 생성자이다.

    public void findColor( ) {
        System.out.println(color + " 공이다.");
    }                              // Ball 클래스에
                                   // 추가한 메서드이다.

    public void findVolume( ) {
        System.out.println("부피는 4/3*(π*반지름*반지름*반지름)이다.");
    }
}
```



상속

Practice

예제 6-3. Circle, Ball 클래스 사용

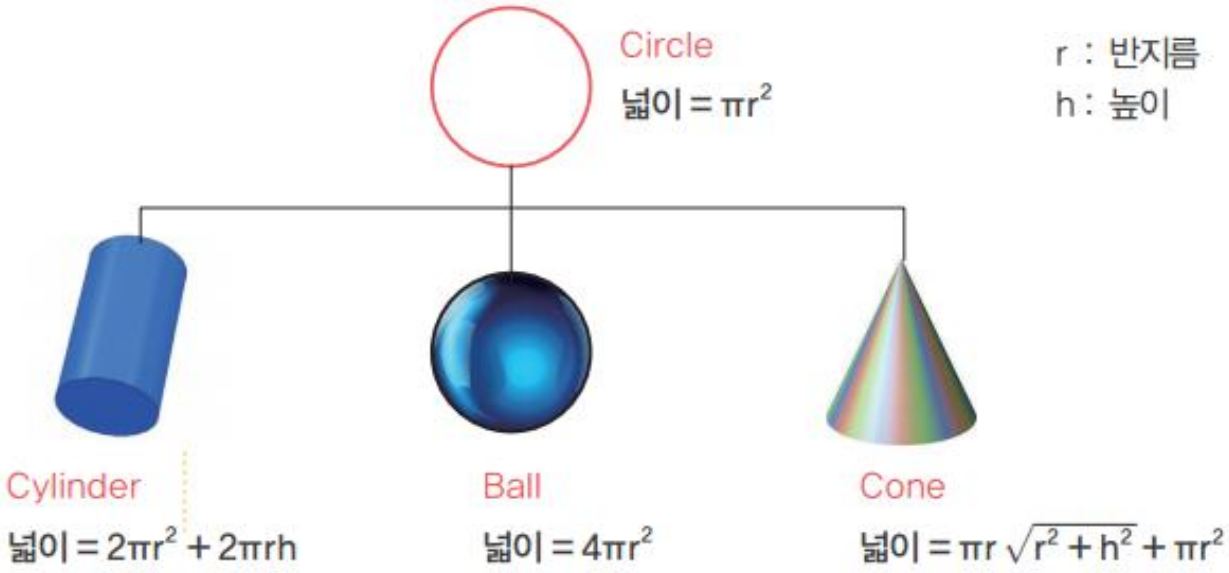
```java
package chap06;

public class InheritanceDemo {
    public static void main(String[] args) {
        Circle c1 = new Circle();
        Ball c2 = new Ball("빨간색");

        System.out.println("원 :");
        c1.findRadius();
        c1.findArea();

        System.out.println("\n공 :");
        c2.findRadius();
        c2.findColor();
        c2.findArea();
        c2.findVolume();
    }
}
```

메서드 오버라이딩

메서드 오버라이딩 (method overriding)

- 물려받은 메서드를 자식 클래스에게 맞도록 수정하는 것

  - 예: 도형의 넓이를 구하는 findArea() 메서드



메서드 오버라이딩

메서드 오버라이딩 규칙

- 부모 클래스의 메서드와 동일한 시그너처를 사용한다. 심지어 반환 타입까지 동일해야 한다.
- 부모 클래스의 메서드보다 접근 범위를 더 좁게 수정할 수 없다.
- 추가적인 예외(Exception)가 발생할 수 있음을 나타낼 수 없다.

메서드 오버라이딩

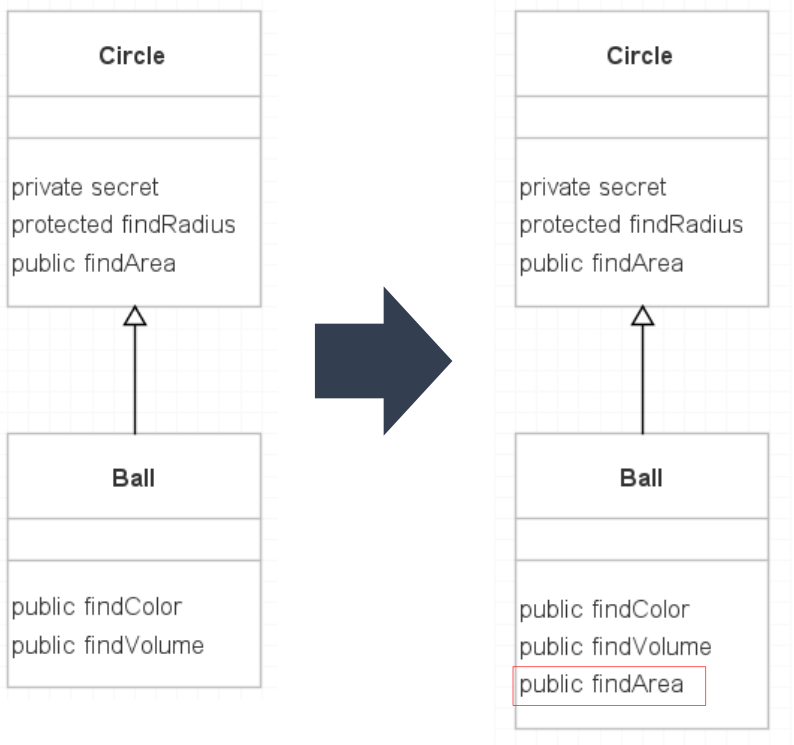
메서드 오버라이딩이 불가한 경우

- private 메서드 : 부모 클래스 전용이므로 자식 클래스에 상속되지 않는다.
- 정적 메서드 : 클래스 소속이므로 자식 클래스에 상속되지 않는다.
- final 메서드 : final 메서드는 더 이상 수정할 수 없으므로 자식 클래스가 오버라이딩할 수 없다.

메서드 오버라이딩

Practice

- 예제 6-4: findArea() 오버라이딩
  - chap6.Ball




```java
package chap06;

public class Ball extends Circle {
    private String color;

    public Ball(String color) {
        this.color = color;
    }

    public void findColor() {
        System.out.println(color + " 공이다.");
    }

    public void findArea() {
        System.out.println("넓이는 4*(π*반지름*반지름)이다.");
    }           // Circle 클래스에서 물려받은 findArea()가 Ball 클래스에는 적합하지
                // 않다. 따라서 자신에게 적합한 메서드로 오버라이딩한다.
    public void findVolume() {
        System.out.println("부피는 4/3*(π*반지름*반지름*반지름)이다.");
    }
}           // @Override 어노테이션
            @Override
            Void findArea() {}
```

메서드 오버라이딩

부모 클래스의 멤버 접근

- 자식 클래스가 메서드를 오버라이딩하면 자식 객체는 부모 클래스의 오버라이딩된 메서드를 숨긴다.
- 그 숨겨진 메서드를 호출하려면 super 키워드를 사용한다.
- super는 현재 객체에서 부모 클래스의 참조를 의미

메서드 오버라이딩

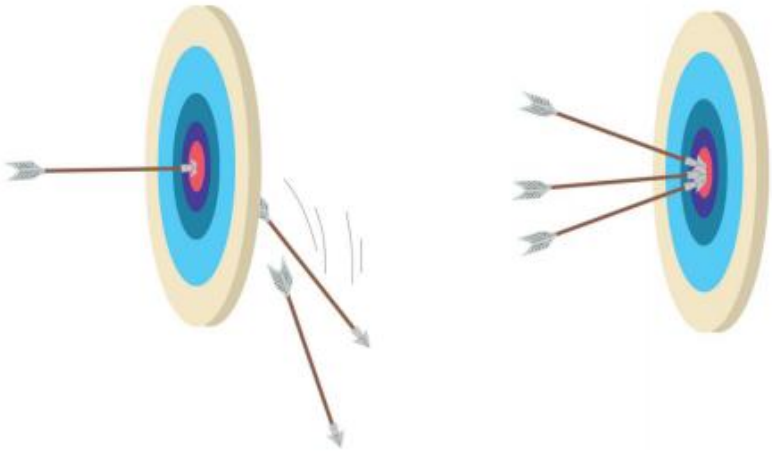
Practice

- 예제 6-5: super 적용
  Circle PPT p. 10

```java
package chap06;

public class Ball extends Circle {
    private String color;

    public Ball(String color) {
        this.color = color;
    }

    public void findColor() {
        System.out.println(color + " 공이다.");
    }

    public void findArea() {
        findRadius(); // <-- super 키워드 없이도 부모 클래스의 메서드 호출 가능

        super.findArea(); // <-- 부모 클래스의 findArea() 호출

        // super.secret(); <-- 부모의 secret 메서드가 private이므로 접근 불가

        System.out.println("넓이는 4*(π*반지름*반지름)이다.");
    }

    public void findVolume() {
        System.out.println("부피는 4/3*(π*반지름*반지름*반지름)이다.");
    }
}
```

메서드 오버라이딩

메서드 오버라이딩과 오버로딩

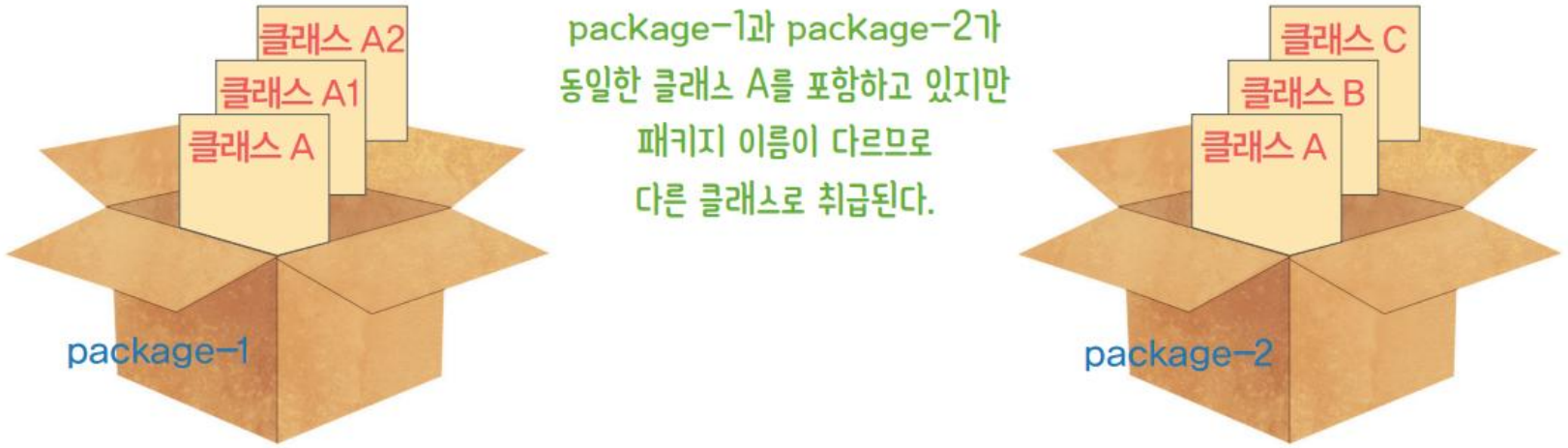


(a) 메서드 오버라이딩　　(b) 메서드 오버로딩

비교 요소	메서드 오버라이딩	메서드 오버로딩
메서드 이름	동일하다.	동일하다.
매개변수	동일하다.	다르다.
반환 타입	동일하다.	관계없다.
상속 관계	필요하다.	필요 없다.
예외와 접근 범위	제약이 있다.	제약이 없다.
바인딩	호출할 메서드를 실행 중 결정하는 동적 바인딩이다.	호출할 메서드를 컴파일할 때 결정하는 정적 바인딩이다.

패키지

- 클래스 파일을 묶어서 관리하기 위한 수단으로 파일 시스템의 폴더를 이용
- 패키지에 의한 장점
  - 패키지마다 별도의 이름 공간(Namespace)이 생기기 때문에 클래스 이름의 유일성을 보장.
  - 클래스를 패키지 단위로도 제어할 수 있기 때문에 좀 더 세밀하게 접근 제어



패키지

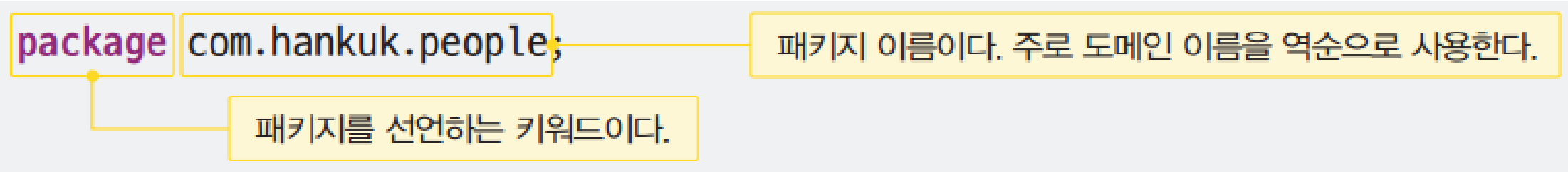
Java에서 제공하는 대표적인 기본 패키지

- java.lang 패키지는 import 문을 선언하지 않아도 자동으로 임포트되는 자바의 기본 클래스를 모아 둔 것
- java.awt 패키지는 그래픽 프로그래밍에 관련된 클래스를 모아둔 것
- java.io 패키지는 입출력과 관련된 클래스를 모아 둔 것

패키지

패키지 선언

- 주석문을 제외하고 반드시 첫 라인에 위치
- 패키지 이름은 모두 소문자로 명명하는 것이 관례. 일반적으로 패키지 이름이 중복되지 않도록 회사의 도메인 이름을 역순으로 사용



패키지

패키지의 사용

- 다른 패키지에 있는 공개된 클래스를 사용하려면 패키지 경로를 컴파일러에게 알려줘야 한다.

com.hankuk.people 패키지

```
public class ShowWorldPeople {
   public static void main(String[] args) {
      com.usa.people.Lincoln greatman
           = new com.usa.people.Lincoln();
   }
}
```

패키지 이름을 접두어로 사용해
다른 패키지에 있는 클래스를 이용한다.

com.usa.people 패키지

```
public class Lincoln { }
```

패키지

import 문

- 패키지의 경로를 미리 컴파일러에게 알려주는 문장
- import 문은 소스 파일에서 package 문과 첫 번째 클래스 선언부 사이에 위치

```
import 패키지이름.클래스;

또는

import 패키지이름.*;
```

```
import com.hankuk.*;         // com.hankuk 패키지에 포함된 모든 클래스이다.
import com.hankuk.people.*; // com.hankuk.people 패키지에 포함된 모든 클래스이다.
```

패키지

import 문

- 예제

```
package com.hankuk.people;

import com.usa.people.Lincoln;

public class ShowWorldPeople {
   public static void main(String[] args) {
      Lincoln greatman = new Lincoln();
   }
}
```

컴파일러에 Lincoln 클래스의 경로를 알려 준다.

import 문으로 경로를 알려 주었으므로 com.usa.people 이라는 경로 정보는 필요 없다.

패키지

정적 import 문

- 정적 클래스 멤버를 import
- 클래스 이름을 생략하고 다른 클래스의 정적 멤버 사용 가능

패키지

Practice

- 예제 6-7: 정적 import 사용

```java
package chap06;

import static java.util.Arrays.sort;

import java.util.Calendar;

public class StaticImportDemo {
    public static void main(String[] args) {
        int[] data = { 3, 5, 1, 7 };

        sort(data);
        System.out.println(Calendar.JANUARY);
        Calendar.getInstance( );
    }
}
```

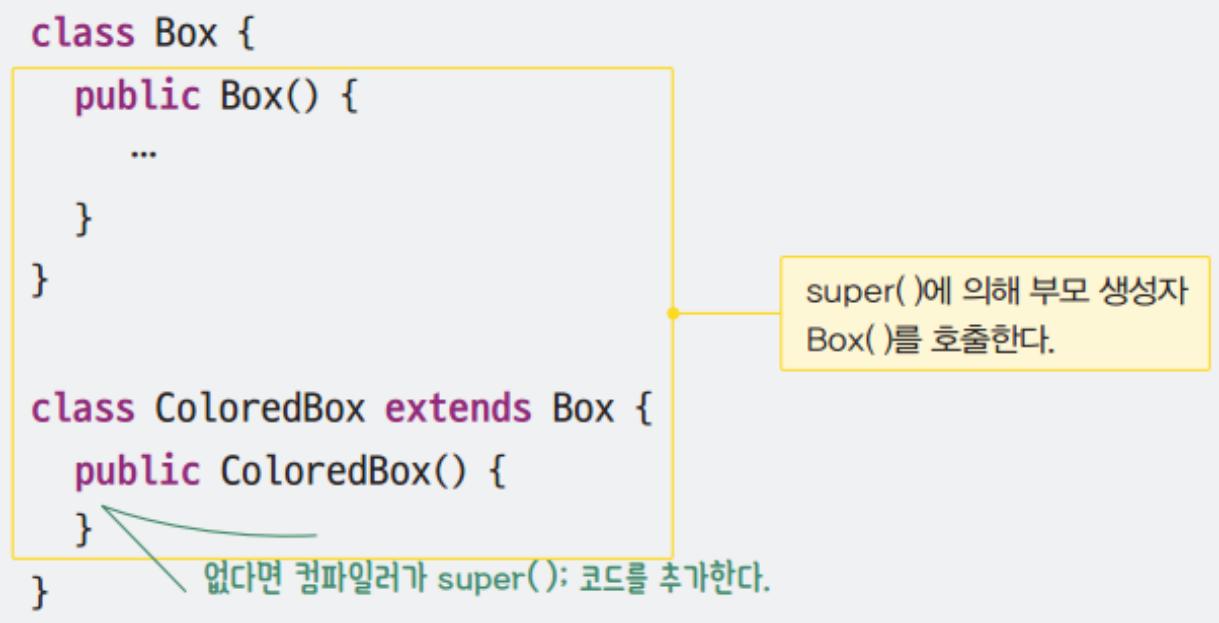
자식 클래스와 부모 생성자

자식 클래스와 부모 생성자

- 자식 생성자를 호출하면
  부모 생성자도 자동으로 호출됨

자식 클래스와 부모 생성자

- 자식 생성자의 첫 행에서 부모의 기본 생성자 호출



자식 클래스와 부모 생성자

- 부모 클래스의 디폴트 생성자가 정의되지 않은 경우
  자식 생성자의 첫 행에 명시적 부모 생성자 호출 필요

```
class Box {
  public Box(String s) {
     …
  }
}

class ColoredBox extends Box {

     // 생성자가 없음

}
```

생성자가 있으므로 컴파일러는 디폴트 생성자 Box()를 추가하지 않는다.

생성자가 없으므로 컴파일러가 디폴트 생성자 ColoredBox()를 추가한다. ColoredBox()는 먼저 부모 생성자 super()를 호출한다. 그런데 부모 클래스에는 Box(String)은 있지만 Box() 생성자는 없어 오류가 발생한다.

자식 클래스와 부모 생성자

Practice

- 예제 6-8: 생성자 실습

```java
package chap06;

class Animal {
    public Animal(String s) {
        System.out.println("동물 : " + s);
    }
}

class Mammal extends Animal {
    public Mammal() {
        // super();
        super("원숭이");
        System.out.println("포유류 : 원숭이");
    }

    public Mammal(String s) {
        super(s);
        System.out.println("포유류 : " + s);
    }
}

public class AnimalDemo {
    public static void main(String[] args) {
        Mammal ape = new Mammal();
        Mammal lion = new Mammal("사자");
    }
}
```