

3부 객체지향

3부에서는 자바 언어의 핵심인 객체지향의 개념과 클래스, 객체, 상속, 다형성 등의 개념과 패키지, 예외처리, 자바의 입출력에 관해 설명합니다.

Chapter 7 : 객체 지향 개념

Chapter 8 : 클래스-속성

Chapter 9 : 클래스-기능

Chapter 10 : 상속

Chapter 11 : 다형성과 추상클래스, 인터페이스

Chapter 12 : 패키지와 `java.lang` 패키지

Chapter 13 : 예외 처리와 입출력



알기 쉽게 해설한 J A V A

9th edition

7장 객체지향 개념

- Section 1 객체 지향의 개요
- Section 2 클래스와 객체
- Section 3 상속
- Section 4 캡슐화
- Section 5 메시지
- Section 6 추상화
- Section 7 다형성





■ 학습목표

9th edition

- 이 장에서는 객체 지향의 5대 핵심 개념에 관해 학습합니다. 이 장에서는 개념에 관해서만 기술합니다. 이러한 개념이 자바 프로그램으로 적용되는 부분은 뒤의 해당 부분에서 자세하게 기술합니다.
- 객체 지향의 개요와 절차 지향과의 차이점을 학습합니다.
- 클래스와 객체를 학습합니다. 객체의 생성 과정을 예제를 통하여 학습합니다.
- 상속의 개념을 예제를 통하여 학습합니다.
- 캡슐화의 개념과 예를 학습합니다.
- 추상화와 다형성의 개념을 학습합니다.

● 객체지향(Object-Oriented)이론

- 컴퓨터를 통하여 실세계와 같은 환경을 흉내(simulation)내기 위해 발전한 이론

● 실세계의 사물 = 속성 + 기능으로 구성

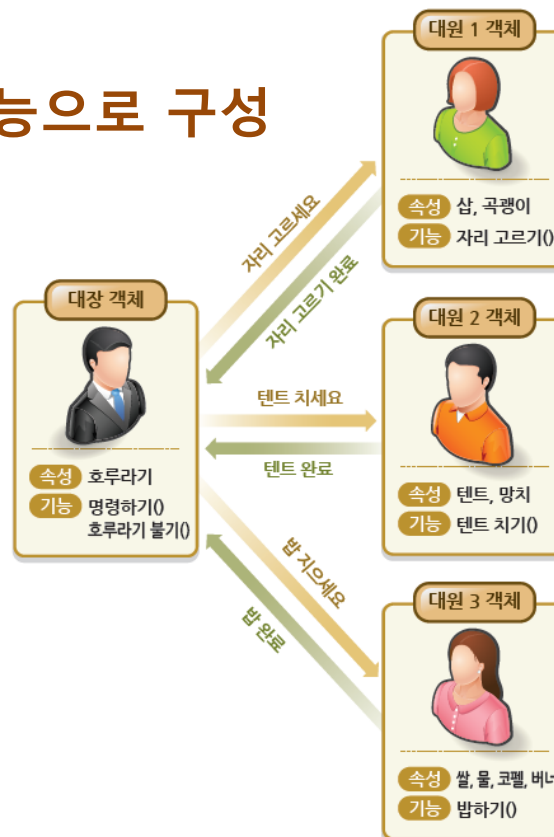


그림 7-1 실제 생활의 문제를 객체 지향적으로 표시

- 객체 지향 이론은 1960년대 클래스, 상속, 캡슐화, 다형성 등의 개념을 중심으로 발전하였으며, 1960년 노르웨이의 달Dahl과 뉘고르Nygaard가 개발한 시물라라는 언어를 최초의 객체 지향 언어라 할 수 있다.

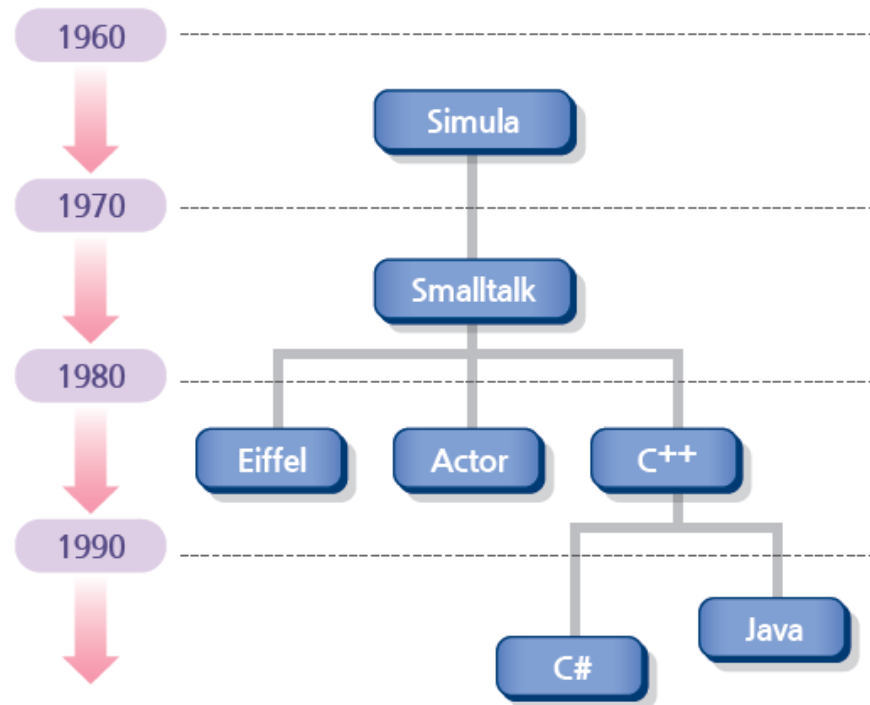


그림 7-2 객체 지향 언어의 역사



1 객체지향의 개요

1-3 객체지향의 장점

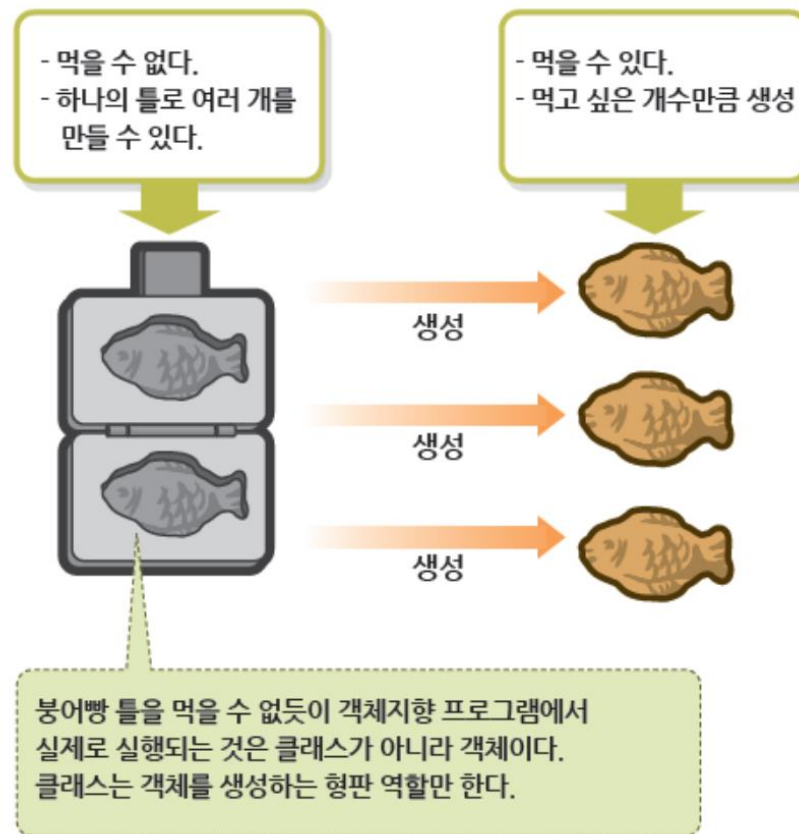
9th edition

● 객체지향의 장점

- 문제를 쉽고 자연스럽게 프로그래밍(모델링) 할 수 있다.
- 쉬운 프로그램의 개발로 인한 생산성을 향상시킬 수 있다.
- 프로그램 모듈을 재사용할 수 있다.

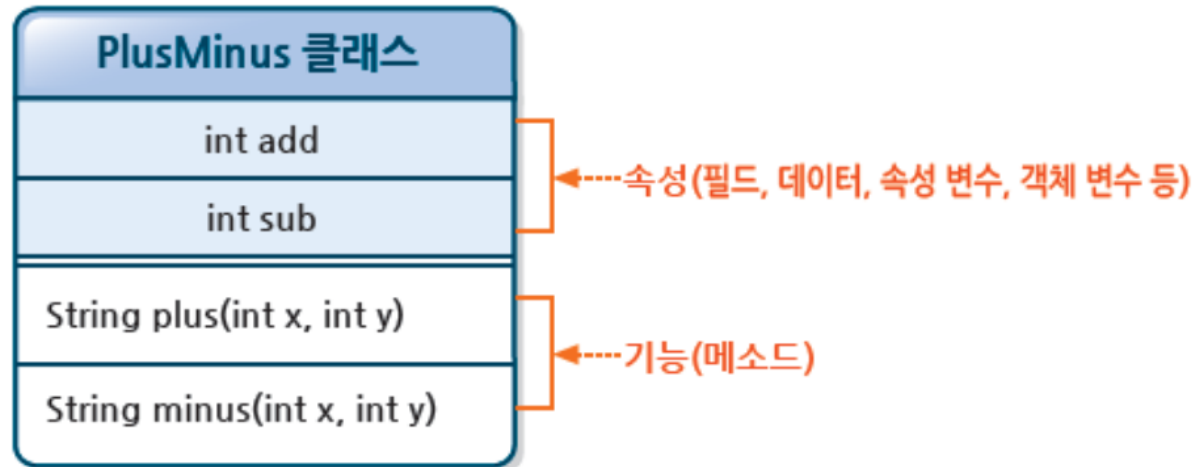
● 클래스

- 하나의 클래스로부터 여러 개의 객체를 생성하기 위해 사용하는 형판



● 객체

- "속성+기능"으로 구성
- 객체를 생성하는 클래스 역시 "속성+기능"으로 구성



예제 7.1

PlusMinus.java

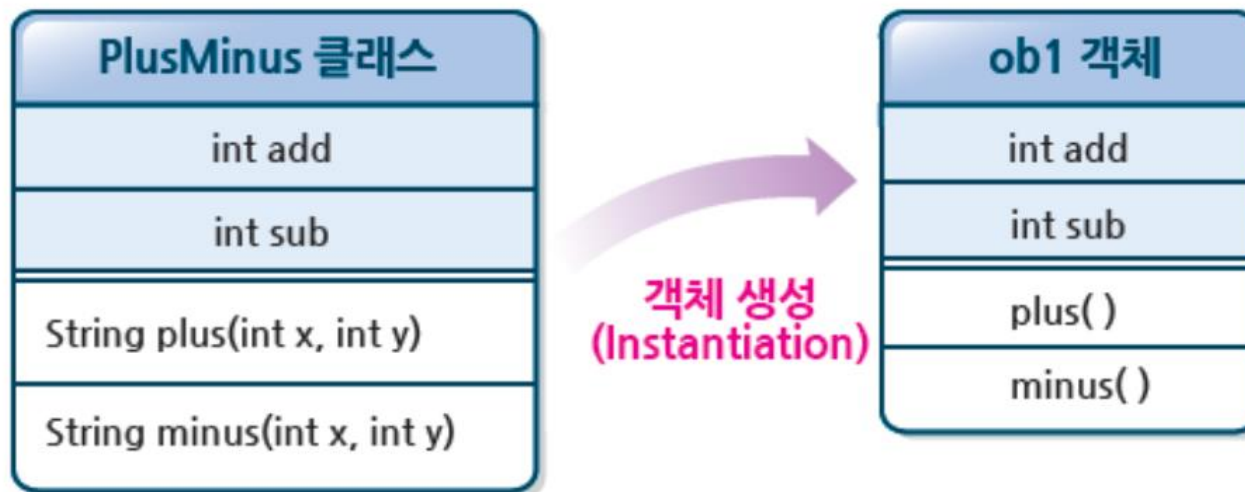
```

01: class PlusMinus {
02:     int add, sub; ← 두 개의 속성을 정의
03:     public String plus(int x, int y) { ←
04:         add = x + y;
05:         return "두 수의 합은 " + plus;
06:     } ←
07:     public String minus(int x, int y) { ←
08:         sub = x - y;
09:         return "두 수의 차는 " + minus;
10:     } ←
11: }
  
```

plus 기능(메소드)을 정의

minus 기능(메소드)을 정의

- 클래스로부터 객체를 생성하는 과정을 실체화(instantiation)라고 하고, 객체를 인스턴스(instance)라 부르기도 합니다. 즉 객체와 인스턴스라는 말은 같은 용어로 간주



예제 7.2

PlusMinusTest1.java

```

01: class PlusMinusTest1 {
02:     public static void main(String args[]) {
03:         PlusMinus ob1 = new PlusMinus(); ← PlusMinus 클래스로부터 ob1 객체 생성
04:         String ssum, sminus;
05:         ssum = ob1.plus(50,30); ←
06:         sminus = ob1.minus(50,30); ←
07:         System.out.println(ssum); ←
08:         System.out.println(sminus); ←
09:     }
10: }

```

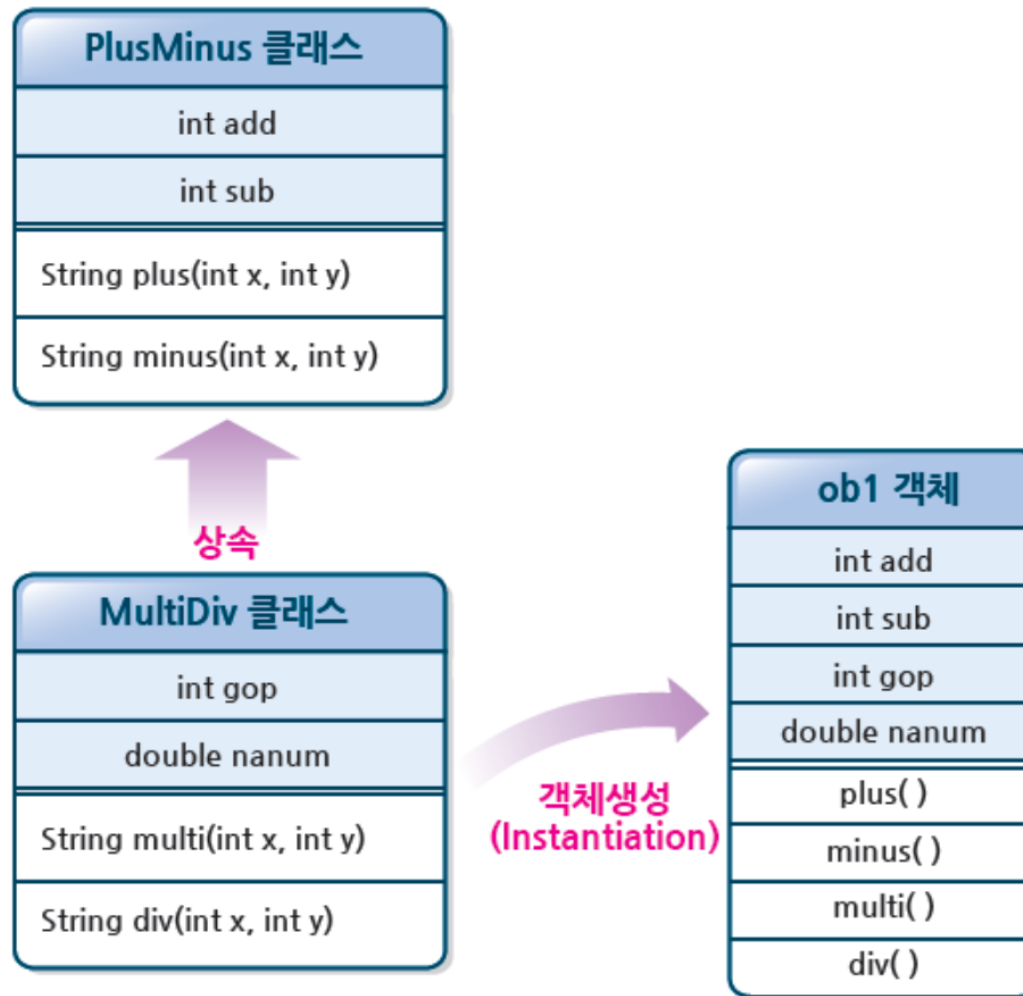
생성된 객체에 메시지를 보내 합과 차를 구한다.

합과 차를 출력

실행 결과

두 수의 합은 80
두 수의 차는 20

- **기존 클래스의 기능을 가지면서 추가적인 기능을 가진 클래스를 만들려면 어떻게 해야 할까요?**
 - 기존 클래스를 그대로 복사하고, 추가적인 기능을 추가하는 방법
 - 이 방법은 코드의 중복이라는 문제와, 추후 중복된 코드가 변경 되었을 때의 복잡한 수정(복사한 곳을 다 수정해야 하는 문제 발생) 문제가 발생.
 - 객체 지향에서는 이러한 문제를 상속이라는 기능으로 해결
 - 즉 새로운 클래스를 만들 때 상위 클래스를 지정함으로써 상위 클래스의 모든 속성과 기능을 상속 받고, 자신의
- **상속의 개념은 확장(extend)의 개념으로 상위클래스를 그대로 상속받고 추가로 확장되는 개념**



예제 7.3

FourRulesTest1.java

```

01: class MultiDiv extends PlusMinus {
02:     int gop;
03:     double nanum;
04:     public String multi(int x, int y) {
05:         gop = x * y;
06:         return "두 수의 곱은 " + gop;
07:     }
08:     public String div(int x, int y) {
09:         nanum = (double) x / y;
10:         return "두 수의 나눈 값은 " + nanum;
11:     }
12: }
13: public class FourRulesTest1 {
14:     public static void main(String args[]) {
  
```

← MultiDiv 클래스의 상위 클래스로 PlusMinus 클래스 지정
 ← 두 개의 속성을 지정
 ← multi 메소드 지정
 ← div 메소드 지정

```

15:    String splus, sminus, smulti, sdiv;
16:    MultiDiv ob1 = new MultiDiv(); ←----- MultiDiv 클래스로부터 객체 생성
17:    splus = ob1.plus(50,30); ←-----
18:    sminus = ob1.minus(50,30); ←----- 객체의 메소드 호출
19:    smulti = ob1.multi(50,30); ←-----
20:    sdiv = ob1.div(50,30); ←-----
21:    System.out.println(splus); ←-----
22:    System.out.println(sminus); ←----- 결과 출력
23:    System.out.println(smulti); ←-----
24:    System.out.println(sdiv); ←-----
25:    }
26: }
  
```

실행 결과

두 수의 합은 80
 두 수의 차는 20
 두 수의 곱은 1500
 두 수의 나눈 값은 1.6666666666666667

● 클래스의 상속은 확장(extend)의 개념으로 계층 구조를 가질 수 있다

- 상위 계층으로 갈수록 공통점은 일반화되고 간단해진다. 하위 계층으로 갈수록 클래스는 특수화되고 개별화된다.

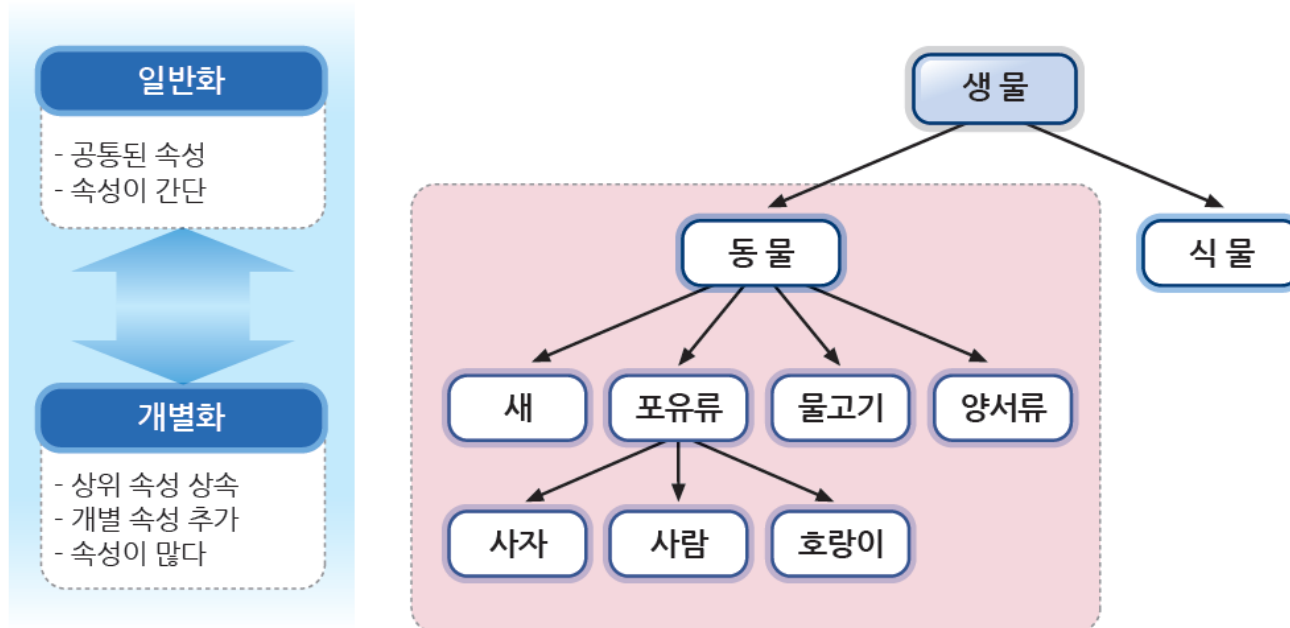
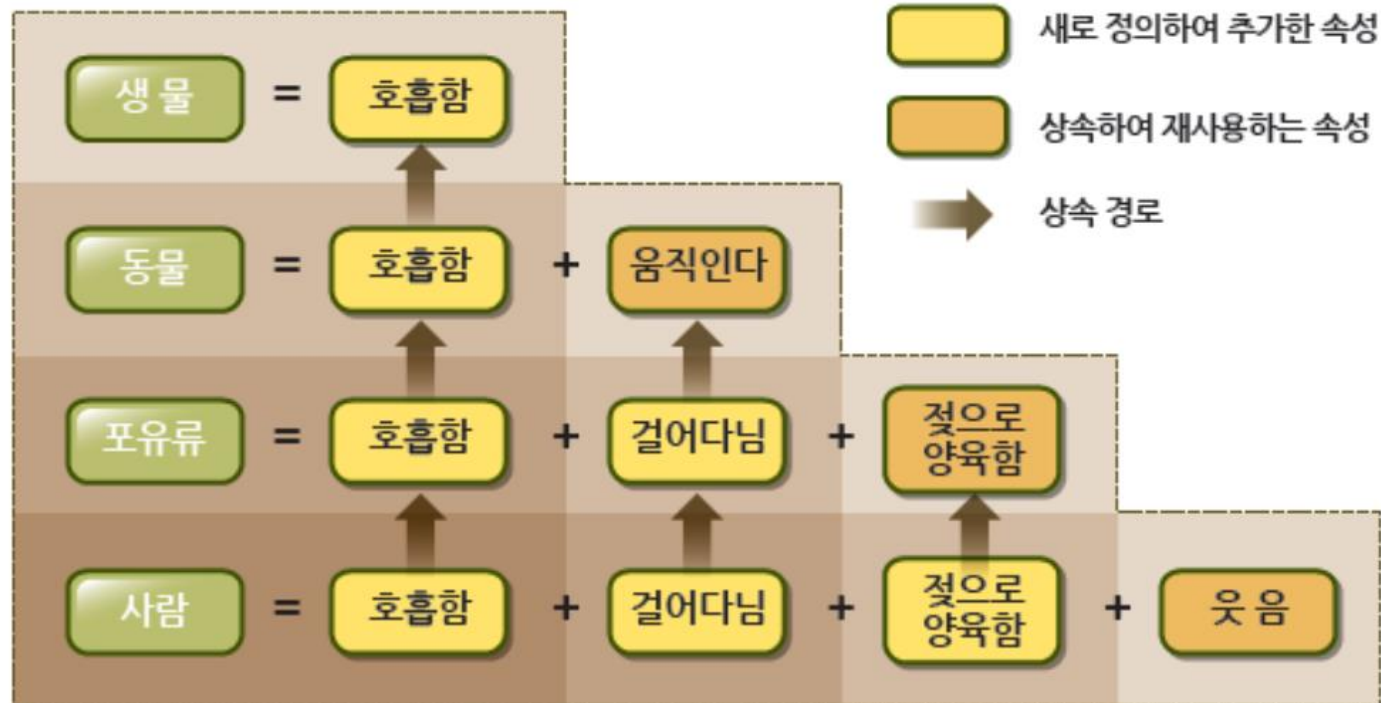


그림 7-9 클래스의 계층 구조

● 클래스 계층 구조에서 상속의 예



● 클래스 상속의 이점

- 소프트웨어 설계를 간단하게 할 수 있는 장점
- 코드를 간결하게 할 수 있다
- 코드의 재 사용성을 높인다

● 캡슐화

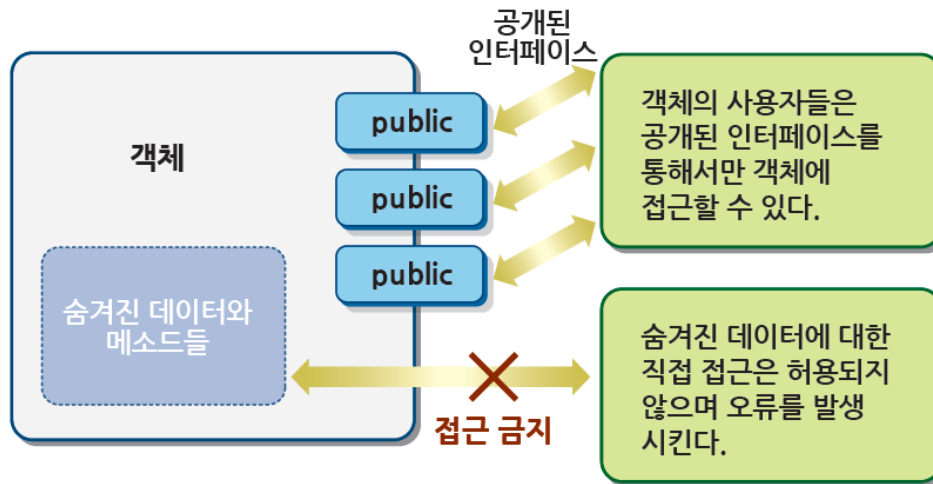
- 감기에 걸렸을 때 먹는 캡슐약과 같은 개념
- 캡슐 약에는 많은 성분이 포함되어 있지만, 단순히 감기를 낫게 해 준다고 생각



사용자 입장에서 약의 자세한 성분이나 제조 방법은 알 필요가 없다.
제약회사 입장에서 약의 성분이나 제조 방법은 보안이 되어야 하는 부분이다.

● 캡슐화를 통한 '정보의 은폐(information hiding)'의 장점

- 객체에 포함된 정보의 손상과 오용을 막을 수 있다.
- 객체 내부의 조작 방법이 바뀌어도 사용방법은 바뀌지 않는다.
- 데이터가 바뀌어도 다른 객체에 영향을 주지 않아 독립성이 유지된다.
- 처리된 결과만 사용하므로 객체의 이식성이 좋다.
- 객체를 부품화 할 수 있어 새로운 시스템의 구성에 부품처럼 사용할 수 있다.



```

01 class MultiDiv extends PlusMinus {
02     int multi; ←----- 속성을 공개(한정자를 지정하지 않았으므로) 모드로 지정
03     private double div; ←----- 속성을 숨김 모드로 지정
04     public String multi(int x, int y) {
05         multi = x * y;
06         return "두 수의 곱은 " + multi;
07     }
08     public String div(int x, int y) {
09         div = (double) x / y;
10         return "두 수의 나눈 값은 " + div;
11     }
12 }

13 public class FourRulesTest1 {
14     public static void main(String args[]) {
15         String plus, minus, multi, div;
16         MultiDiv ob1 = new MultiDiv();
17         .....
18         ob1.multi = 200; ←----- 공개된 속성이므로 속성값을 직접 조정 가능
19         ob1.div = 3.6666; ←----- 숨김 속성에 접근하므로 오류가 발생
20         .....
  
```

● 메시지

- 객체에 일을 시키는 행위

● 객체 사이의 메시지 전달

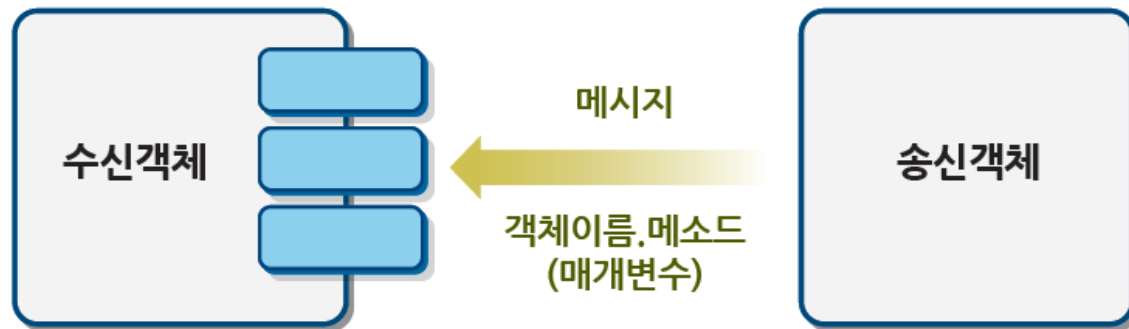


그림 7-13 객체 사이의 메시지 전달

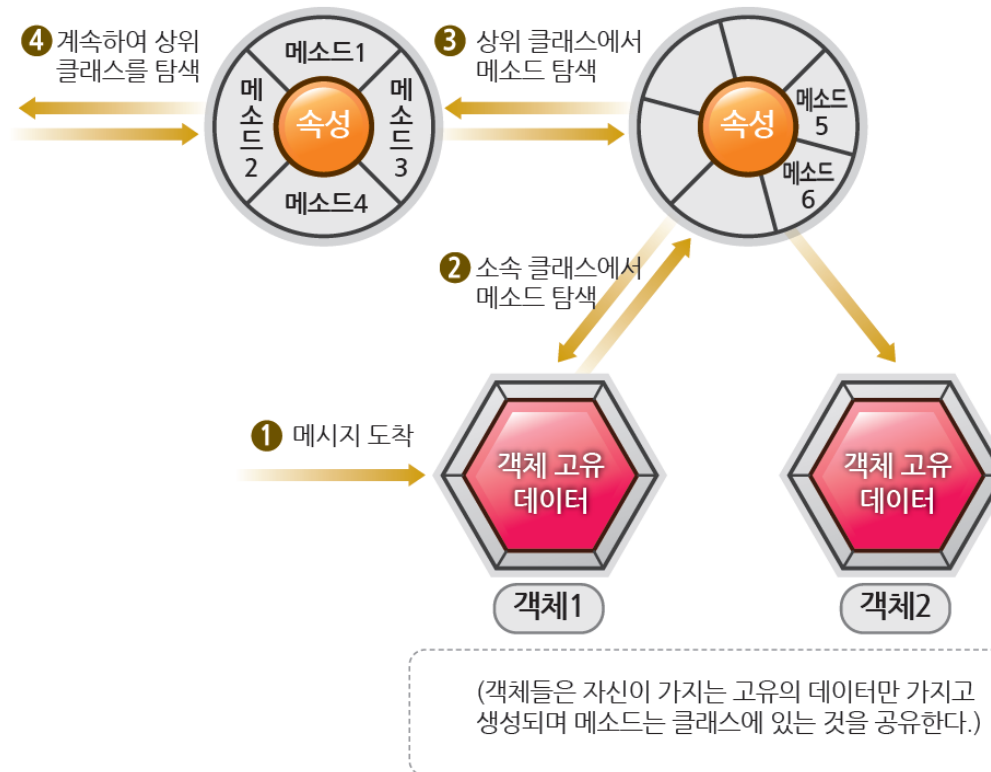
● 메시지의 예

```
01 public class FourRulesTest1 {  
02     public static void main(String args[]) {  
03         String plus, minus, multi, div;  
04         MultiDiv ob1 = new MultiDiv();  
05         plus = ob1.plus(50,30);  
06         minus = ob1.minus(50,30);  
07         multi = ob1.multi(50,30);  
08         div = ob1.div(50,30);  
09         .....  
10     }  
11 }
```

ob1 객체를 통하여 수행할 수 있는 메소드를 호출하는 메시지. 메시지의 구성은 객체 이름.메소드 이름(매개 변수)으로 구성된다.

● 상속된 구조에서의 메소드 탐색

- 최 상위 클래스에도 탐색하고자 하는 메소드가 없는 경우 오류 발생



● 실세계의 문제를 객체로 전환할 때 중요한 개념

- 추상화는 복잡한 문제들 중에 공통적인 부분을 추출하여 추상 클래스로 제공하고, 상속을 이용하여 나머지 클래스들을 하위 클래스로 제공하는 기법

▣ 추상화의 단계

- 1단계 : 현실 세계의 문제들이 가지는 공통적인 속성을 추출
- 2단계 : 공통 속성을 가지는 추상 클래스 작성
- 3단계 : 추상 클래스의 하위 클래스로 현실 세계의 문제들을 구현

● 추상화를 하지 않을 경우에 도형을 그리는 문제

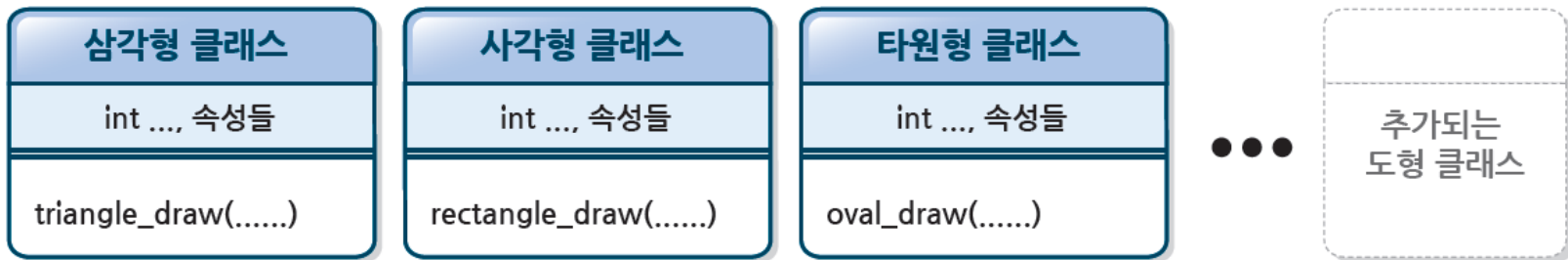
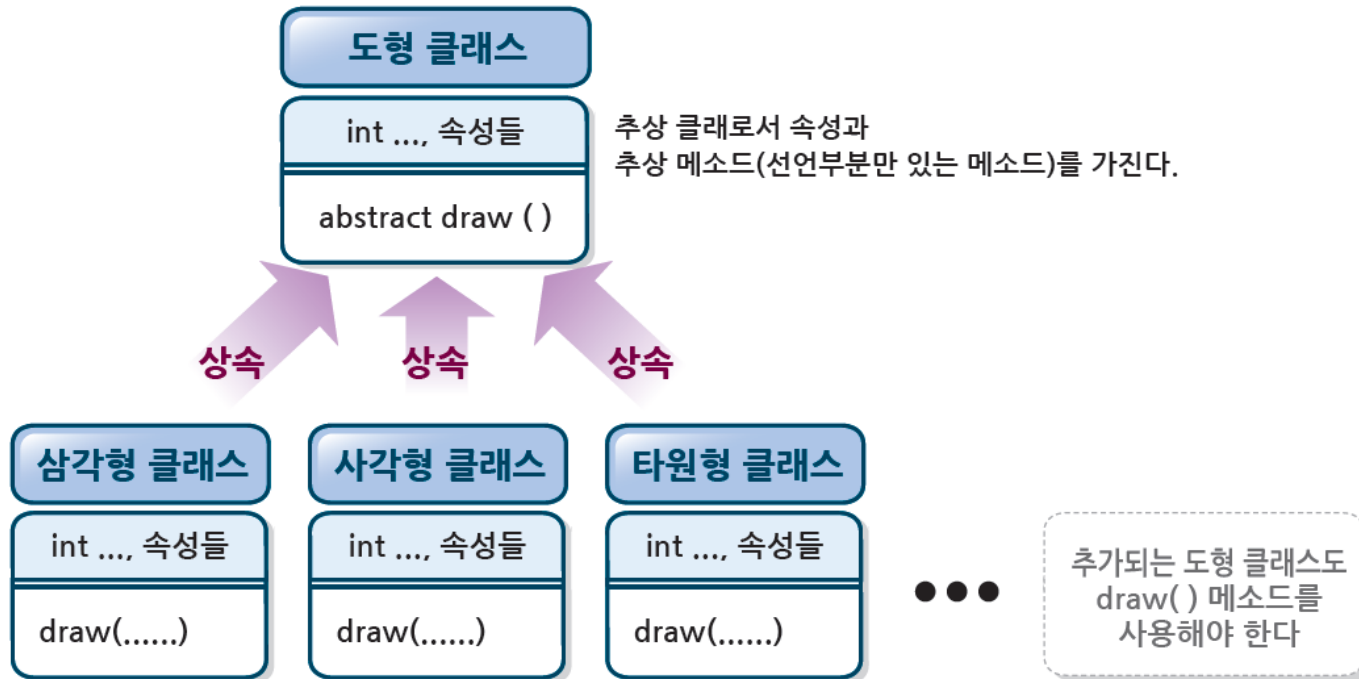


그림 7-15 클래스를 연관시키지 않고 작성

● 추상화를 적용하여 도형을 그리는 문제



추상 클래스의 하위 클래스는
추상 클래스에 선언된 추상 메소드를 치환하여 사용해야 한다.

그림 7-16 추상 클래스를 사용하여 체계적으로 클래스를 구성

● 다형성(Polymorphism)

- 객체지향의 중요한 개념 중에 하나로서 다양한(poly) 변신(morphism)을 의미
- 서로 다른 객체가 동일한 메시지에 대하여 서로 다른 방법으로 응답할 수 있는 기능

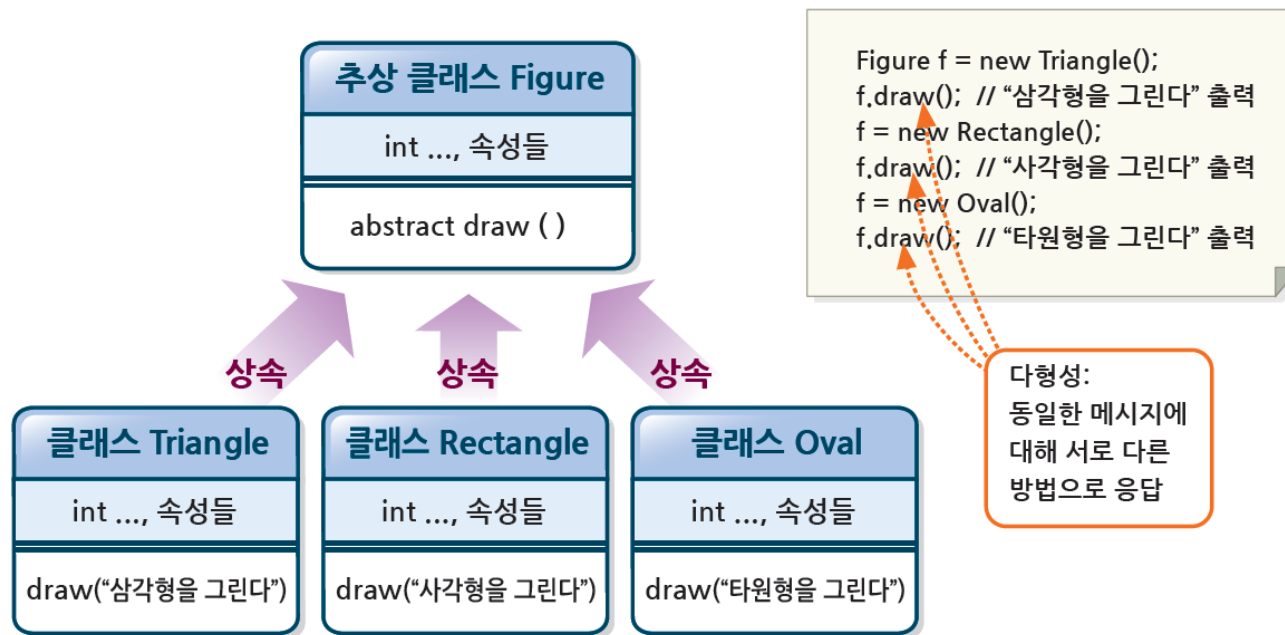


그림 7-17 다형성의 개념

● 객체 지향의 개요

- ① 객체 지향은 실세계의 요소들을 “속성+기능”으로 표현한 것입니다.
- ② 자바는 객체 지향의 역사적 측면에서 볼 때 대표적으로 성공한 언어입니다.
- ③ 절차 지향은 기능 중심과 자료와 절차의 분리에서 오는 문제점을 가진 언어입니다.
- ④ 소프트웨어 개발 분야에서 약 60% 가까이 객체 지향 언어가 사용되고 있습니다.
- ⑤ 객체 지향은 우리들의 실세계를 그대로 반영할 수 있는 프로그래밍 방법입니다.

● 클래스와 객체

- ① 클래스는 객체를 생성하는 형판template 역할을 합니다.
- ② 하나의 클래스로부터 다수 개의 객체가 생성됩니다.
- ③ 클래스는 “속성+기능”으로 구성됩니다.
- ④ 클래스로부터 객체를 생성하는 과정을 실체화instantiation라고 하며, 객체를 인스턴스라 부르기도 합니다.
- ⑤ 객체 지향은 절차 지향과는 달리 프로그램의 기본 단위가 객체(속성+기능)입니다.

● 상속

- ① 상속을 이용하여 비슷한 기능을 가진 클래스들을 계층 구조 형태로 구성할 수 있습니다.
- ② 상속은 상위 클래스의 모든 속성과 기능을 상속받고, 추가로 더 가지는 확장extend의 개념입니다.
- ③ 자바에서는 하나의 클래스만을 상위 클래스로 가질 수 있습니다. 즉 다중 상속을 허용하지 않습니다.
- ④ 클래스의 상속을 나타내는 계층 구조에서 상위로 올라갈수록 일반화된 공통 속성을 가지며 하위로 내려갈수록 개별 클래스가 가지는 개별화가 이루어집니다.

● 캡슐화

- ① 캡슐화는 꼭 필요한 내용만 보여 주고, 불필요한 내용을 감춘다는 개념입니다.
- ② 클래스를 작성할 때는 꼭 필요한 내용만 공개해야 합니다.
- ③ 클래스를 작성할 때 캡슐화 개념을 적용하지 않으면, 클래스가 잘못 사용될 수 있습니다.
- ④ 캡슐화의 개념을 적용하여 정보 은폐Information hiding를 달성할 수 있습니다.

● 메시지

- ① 생성된 객체들 사이의 상호 관계를 정의한 것이 메시지입니다.
- ② 객체들은 메시지를 통하여 상대 객체에게 일을 시킵니다.
- ③ 메시지는 객체 이름, 메소드 이름, 메소드 수행에 필요한 인자argument를 포함합니다.
- ④ 상속 관계의 클래스 구조에서 메소드가 호출되면 자신의 클래스에서 메소드를 탐색하고, 없을 경우 상위 클래스의 메소드를 탐색합니다. 이러한 과정을 반복적으로 거쳐도 메소드가 발견되지 않는 경우는 오류가 발생합니다.

● 추상화

- ① 실세계의 문제를 객체로 변환할 때 적용할 수 있는 개념입니다.
- ② 여러 개의 문제에서 공통적인 속성을 추출하여 상위 클래스에 작성하고, 나머지 클래스를 하위 클래스로 구성하는 것이 추상화의 핵심입니다.
- ③ 추상화와 상속은 다형성을 제공하는 기반입니다.

● 다형성

- ① 다형성은 다양한 변신을 의미하는 개념입니다.
- ② 다형성은 서로 다른 객체가 동일한 메시지에 대하여 서로 다른 방법으로 응답할 수 있는 기능으로 정의할 수 있습니다.
- ③ 다형성을 구현하기 위해서는 상속과 추상 클래스를 이용합니다.