Chapter 11. 다형성과 추상 클래스, 인터페이스

학습목표

- 다형성을 위한 객체의 형 변환에 대해 학습합니다.
- 연산자 instanceof에 대해 학습합니다.
- 다형성의 장점과 구현 방법을 학습합니다.
- 추상 클래스의 개념과 얻어지는 장점, 추상 클래스와 추상 메소드에 관해 학습합니다.
- 추상 클래스와 객체의 형 변환, 오버라이딩을 활용한 다형성을 학습합니다.
- 인터페이스의 개념과 상속, 사용 방법에 관해 학습합니다.
- 인터페이스를 활용한 다형성을 학습합니다.

목차

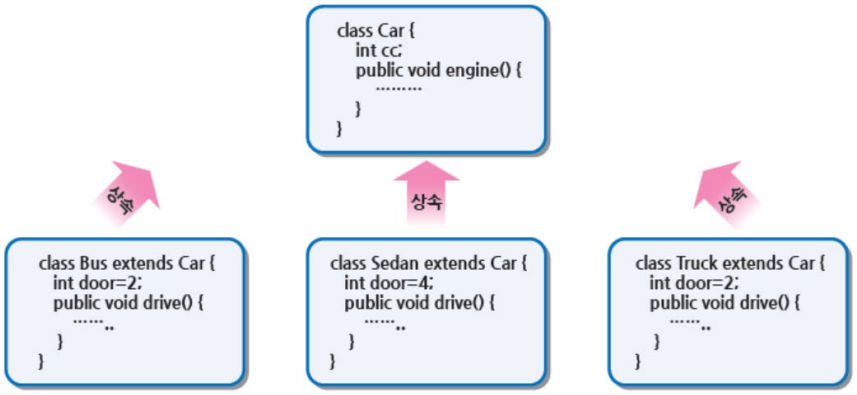
Section 1.

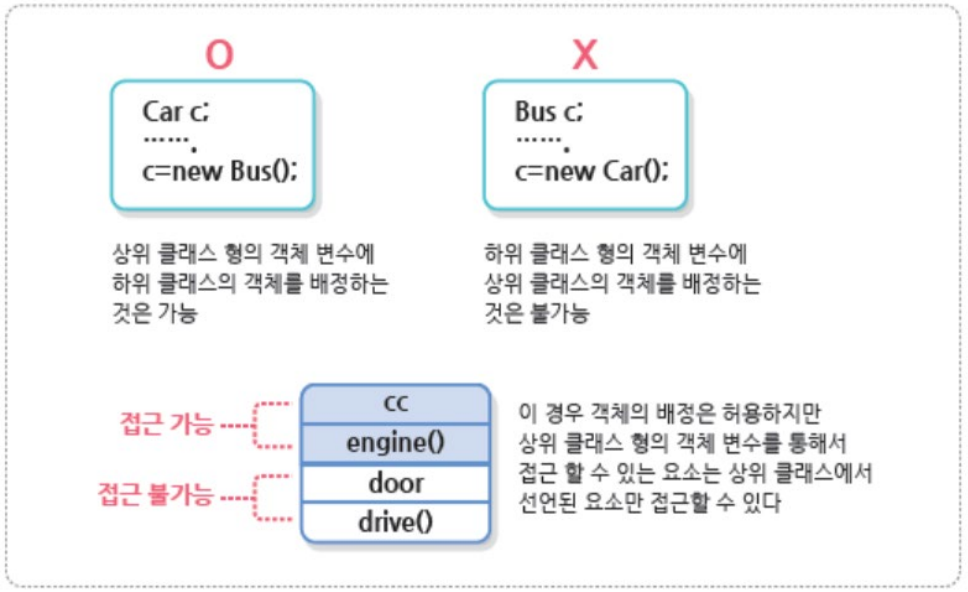
객체의 형 변환

1 객체의 형 변환

● 상속된 관계에서 객체의 형 변환을 허용

- 상속된 관계에서 상위 클래스 형의 객체 변수에 하위 클래스에서 생성된 객체의 배정을 허용
- 반대로 하위 클래스 형의 객체 변수에 상위 클래스에서 생성된 객체의 배정은 허용하지 않는다





1 객체의 형 변환

Object 클래스는 자바의 최상위 클래스

```
01: Object o1 = new Scanner();
02: Object o2 = new Box();
03: Scanner s = new Object();
04: Random r = new Object();
```

01, 02: 가능. 단 이 객체 변수를 통해서는 Object 클래스의 속성과 기능만 사용이 가능

03, 04: 오류 발생. 상위 클래스의 객체를 배정할 수 없다.

결론적으로 상속 관계에 있는 클래스 사이에서 객체의 형 변환은 다음과 같습니다.

- **하위 클래스에서 생성된 객체를 상위 클래스 형의 객체 변수에 배정하는 형 변환은 허용**합니다.
- 반대로 **상위 클래스에서 생성된 객체를 하위 클래스 형의 객체 변수에 배정할 수 없습니다.**
- 상위 클래스 형의 객체 변수에 배정된 하위 클래스 객체의 경우, **상위 클래스 형의 객체 변수를 통해서는 상위 클래스에 선언된 속성과 기능에만 접근이 가능**합니다.

1 객체의 형 변환

- 예제 11.1

```
01: class Rectangle4 {
02:     public int width;
03:     public int height;
04:     public Rectangle4(int w, int h) {
05:         width = w;
06:        height = h;
07:     }
08:     public int computeRectangleArea() {
09:         return width * height;
10:     }
11: }
12: class Cube4 extends Rectangle4 {
13:     public int depth;
14:     public Cube4(int w, int h, int d) {
15:         super(w,h);
16:         depth = d;
17:     }
18:     public int computeCubeArea() {
19:         return super.computeRectangleArea() * depth;
20:     }
21: }
22: public class CastTest1 {
23:     public static void main(String args[]) {
24:         Rectangle4 r = new Cube4(10,20,30);
25:         System.out.println("넓이는 : "+r.computeRectangleArea());
26:         //System.out.println("넓이는 : "+r.computeCubeArea());
27:         //Cube4 c = new Rectangle4(10,20);
28:     }
29: }
```

super를 활용하여 상위 클래스의 생성자와 메소드 호출

상위 클래스 형의 객체 변수에 하위 클래스의 객체를 생성하여 배정

객체 변수를 통하여 상위 클래스의 메소드 호출

오류 발생 하위 클래스의 메소드 호출은 불가능

하위 클래스의 객체 변수에 상위 클래스의 객체 배정 불가능

실행 결과

```
넓이는 : 200
```

1 객체의 형 변환

자바의 기본 자료형에서는 값의 손실이 발생하는 경우에도 명시적인 형 변환을 허용한다고 설명하였습니다(3장 참조). 그러나 자바의 클래스에서는 하위 클래스의 객체 변수에 상위 클래스에서 생성된 객체를 배정하는 것을 허용하지 않습니다. 즉 기본 자료형과는 달리 참조 자료형에서는 확대 형 변환은 가능하나, 축소 형 변환은 명시적인 형 변환 구문을 적용하여도 불가능합니다.

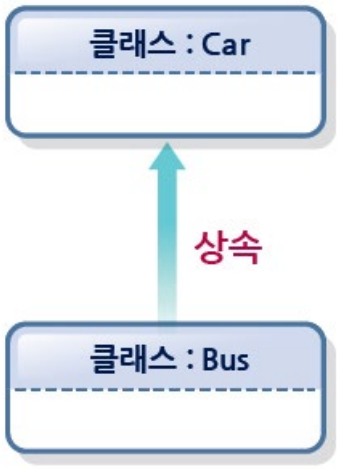


그림 11-2 상속관계

```
01: class Car {....}
02: class Bus extends Car {....}
03: public class Casting {
04:     .....
05:     Car c1;
06:     Bus b1 = new Bus();
07:     c1 = b1; // c1 = (Car)b1;<-------- 형 변환 가능(확대 형 변환이기 때문에 명시적인 형 변환 생략 가능)
08:     Car c2 = new Car();
09:     Bus b2 = (Bus)c2; <---------- 실행시간 오류. 상속 관계에서 축소 형 변환은 불가능
10:     Car c3 = new Bus(); <-------- 상위 클래스의 객체 변수에 하위 클래스의 객체 배정 가능
11:     Bus b3 = (Bus)c3; <---------- c3에 배정된 객체가 Bus 클래스의 객체이므로 명시적인 형 변환 가능
12: }
```

1 객체의 형 변환

예제 11.2 CastTest2.java

```
01: class Rectangle5 {
02:     public int width;
03:     public int height;
04:     public Rectangle5(int w, int h) {
05:         width = w;
06:         height = h;
07:     }
08:     public int computeRectangleArea() {
09:         return width * height;
10:     }
11: }
12: class Cube5 extends Rectangle5 {
13:     public int depth;
14:     public Cube5(int w, int h, int d) {
15:         super(w,h);
16:         depth = d;
17:     }
18:     public int computeCubeArea() {
19:         return super.computeRectangleArea() * depth;
20:     }
21: }
22: public class CastTest2 {
23:     public static void main(String args[]) {
24:         Rectangle5 r = new Cube5(10,20,30);
25:         System.out.println("정사각형의 넓이는 : "+r.computeRectangleArea());
26:         //System.out.println("넓이는 : "+r.computeCubeArea());
27:         Cube5 c = (Cube5) r;
28:         System.out.println("직육면체의 부피는 : "+c.computeCubeArea());
29:     }
30: }
```

상위 클래스 형의 객체 변수에 하위 클래스의 객체를 생성하여 배정

객체 변수를 통하여 상위 클래스의 메소드 호출

하위 클래스로 형 변환을 시도하여 하위 클래스 객체 변수에 배정

하위 클래스 메소드 호출 가능

오류 발생 하위 클래스의 메소드 호출은 불가능

실행 결과

```
정사각형의 넓이는 : 200
직육면체의 부피는 : 6000
```

Section 2.

연산자 instanceof

2 연산자 instanceof

- **instanceof 연산자**
  - 객체가 특정 클래스로부터 생성된 객체인지를 판별하여 true 또는 false값을 반환

[형식] 연산자 instanceof

```
객체 변수 instanceof Class-Type
```

2 연산자 instanceof

예제 11.3 InstanceTest1.java

```
01: class Rectangle {
02:     public int width;
03:     public int height;
04:     public Rectangle(int w, int h) {
05:         width = w;
06:         height = h;
07:     }
08:     public int computeRectangleArea() {
09:         return width * height;
10:     }
11: }
12: class Cube extends Rectangle {
13:     public int depth;
14:     public Cube(int w, int h, int d) {
15:         super(w,h);
16:         depth = d;
17:     }
18:     public int computeCubeArea() {
19:         return super.computeRectangleArea() * depth;
20:     }
21: }
```

2 연산자 instanceof

```
22: public class InstanceTest1 {
23:     public static void main(String args[]) {
24:         Rectangle r = new Rectangle(10,20);
25:         Cube c = new Cube(10,20,30);
26:         System.out.println("r이 Rectangle의 객체? : "+ (r instanceof
    Rectangle));
27:         System.out.println("r이 Cube의 객체? : "+ (r instanceof Cube));
28:         System.out.println("c가 Rectangle의 객체? : "+ (c instanceof
    Rectangle));
29:         System.out.println("c가 Cube의 객체? : "+ (c instanceof Cube));
30:         System.out.println("=========형 변환 이후==========");
31:         r = new Cube(20,30,40);
32:         System.out.println("형 변환 r이 Rectangle의 객체? : "+ (r
    instanceof Rectangle));
33:         System.out.println("형 변환 r이 Cube의 객체? : "+ (r instanceof
    Cube));
34:         System.out.println("형 변환 r이 Rectangle의 객체? : "+ (c
    instanceof Rectangle));
35:         System.out.println("형 변환 r이 Cube의 객체? : "+ (c instanceof
    Cube));
36:         System.out.println("============================");
37:         System.out.println("c가 Object의 객체? : "+ (c instanceof Object));
38:   }
39: }
```

실행 결과

```
r이 Rectangle의 객체? : true
r이 Cube의 객체? : false
c가 Rectangle의 객체? : true
c가 Cube의 객체? : true
=========형 변환 이후==========
형 변환 r이 Rectangle의 객체? : true
형 변환 r이 Cube의 객체? : true
형 변환 r이 Rectangle의 객체? : true
형 변환 r이 Cube의 객체? : true
============================
c가 Object의 객체? : true
```

2 연산자 instanceof

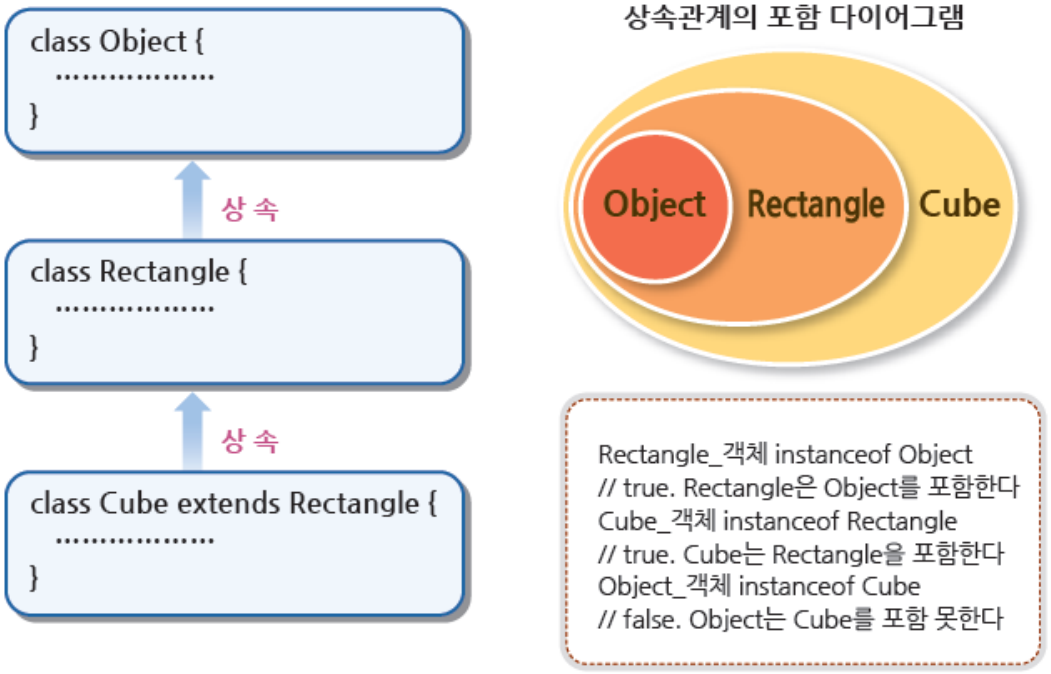


그림 11-2 상속 관계와 포함 다이어그램

Section 3.

다형성

3 다형성

- **객체 지향에서 다형성은 “서로 다른 객체가 동일한 메시지에 대하여 서로 다른 방법으로 응답할 수 있는 기능”이라고 정의할 수 있다**
  - 다형성은 앞 장에서 기술한 상속과 객체의 형 변환, 메소드 오버라이딩을 통하여 구현될 수 있다.

3 다형성

다음 프로그램은 세 개의 클래스가 상속관계에 있으면서 하나의 속성과 하나의 메소드를 가지고 있습니다. 객체 지향 언어의 다형성(polymorphism)을 제공하는 예

예제 11.4 PolymorphismTest1.java

```
01: class Am {
02:     int count=1; <---------- count 값을 1로 설정
03:     void callme() {
04:         System.out.println("Am의 callme() 실행, count 값 : "+count);
05:     }
06: }
07: class Bm extends Am {
08:     int count=2; <---------- count 값을 2로 설정
09:     void callme() { <---------- 상위 클래스의 callme() 메소드 오버라이딩
10:         System.out.println("Bm의 callme() 실행, count 값 : "+count);
11:     }
12: }
13:
14: class Cm extends Am {
15:     int count=3; <---------- count 값을 3으로 설정
16:     void callme() { <---------- 상위 클래스의 callme() 메소드 오버라이딩
17:         System.out.println("Cm의 callme() 실행, count 값 : "+count);
18:     }
19: }
```

3 다형성

```
20:
21: class PolymorphismTest1 {
22:     public static void main(String args[]) {
23:         Am r = new Am();
24:         r.callme();
25:         System.out.println("r.count 값 : " + r.count);
26:         r = new Bm();
27:         r.callme();
28:         System.out.println("r.count 값 : " + r.count);
29:         r = new Cm();
30:         r.callme();
31:         System.out.println("r.count 값 : " + r.count);
32:     }
33: }
```

Am 클래스로부터 객체 생성력

callme() 호출. Am 클래스의 callme() 수행. count 값으로 1 출력문자열 출력

r.count 출력. Am 클래스의 count 값 1이 출력

Am 클래스 형의 객체변수 r에 Bm 클래스의 객체 배정(형 변환)

callme() 호출. Bm 클래스의 callme() 수행. count 값으로 2 출력

r.count 출력. Am 클래스의 count 값 1이 출력

Am 클래스 형의 객체변수 r에 Cm 클래스의 객체 배정(형 변환)

callme() 호출. Cm 클래스의 callme() 수행. count 값으로 3 출력

r.count 출력. Am 클래스의 count 값 1이 출력

실행 결과

```
Am의 callme() 실행, count 값 : 1
r.count 값 : 1
Bm의 callme() 실행, count 값 : 2
r.count 값 : 1
Cm의 callme() 실행, count 값 : 3
r.count 값 : 1
```

3 다형성

예제 11.5 PolymorphismTest2.java

```
01: class Item {                                  ← price 속성을 가진 Item 클래스 생성
02:     public int price;
03: }
04: class Noodle extends Item {                   ← Item의 하위 클래스로 생성자에서 속성값을 지정
05:     public Noodle() {
06:         price = 2500;
07:     }
08:     public String toString() { return "국수";}  ← Object 클래스의 toString() 메소드 오버라이딩
09: }
10: class Mixnoodle extends Item {                ← Item의 하위 클래스로 생성자에서 속성값을 지정
11:     public Mixnoodle() {                      ← 각 클래스의 객체로 buy() 메소드 호출
12:         price = 3000;
13:     }
14:     public String toString() { return "비빔 국수";} ← Object 클래스의 toString() 메소드 오버라이딩
15: }
16: class Wodong extends Item {                   ← Item의 하위 클래스로 생성자에서 속성값을 지정
17:     public Wodong() {
18:         price = 2500;
19:     }
20:     public String toString() { return "우동";}  ← Object 클래스의 toString() 메소드 오버라이딩
21: }
```

3 다형성

```
22: class Buyer {
23:    private int money;
24:    public Buyer(int money) {
25:         this.money = money;
26:     }
27:     public void buy(Item it, int count) {
28:         System.out.println(it + "을(를) " + count + "개 맛있게 먹었습니다");
29:         money = money - it.price*count;
30:         System.out.println("현재 남은 돈 : " + money);
31:     }
32: }
33: public class PolymorphismTest2 {
34:     public static void main(String args[]) {
35:         Buyer me = new Buyer(20000);
36:         me.buy(new Wodong(),3);
37:         me.buy(new Noodle(),2);
38:         me.buy(new Mixnoodle(),2);
39:     }
40: }
```

buy 메소드의 매개 변수로 Item 클래스 형의 객체를 선언

실행 결과

```
우동을(를) 3개 맛있게 먹었습니다
현재 남은 돈 : 12500
국수을(를) 2개 맛있게 먹었습니다
현재 남은 돈 : 7500
비빔 국수을(를) 2개 맛있게 먹었습니다
현재 남은 돈 : 1500
```

Section 4.

추상 클래스

4 추상 클래스

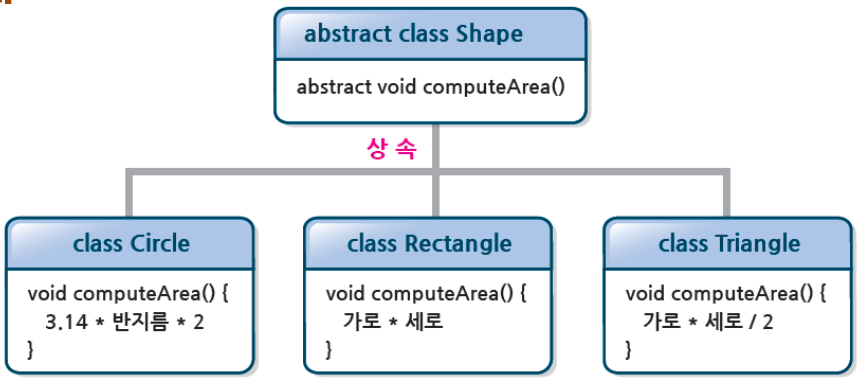
- **추상화는 복잡한 문제들 중에 공통적인 부분을 추출하여 추상 클래스로 제공하고, 상속을 이용하여 나머지 클래스들을 하위 클래스로 제공하는 기법**

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

4 추상 클래스

- 추상화 클래스의 예



- 추상화 클래스와 추상 메소드

[형식] 추상 클래스와 추상 메소드

```
abstract class 클래스 이름 {        ← 추상 클래스(이 클래스로부터 객체를 생성할 수 없다)
    .....
    클래스에 기술할 수 있는 일반적인 멤버 변수와 메소드
    abstract void 추상 메소드 이름();   ← 추상 메소드(선언 부분만 기술한다). 메소드의 몸체
                                        가 없기 때문에 끝에 ";"을 붙여야 한다.
}
```

4 추상 클래스

● 추상화 클래스와 추상 메소드의 예

[예]

```
01: abstract class Shape {          <-- 추상 클래스 선언
02:     .............
03:     abstract void computeArea(); <-- 추상 메소드 선언. 선언부만 기술
04:     .............
05: }
06:
07: public class Circle extends Shape {   <-- 추상 클래스로부터 상속받아 Circle 클래스 생성
08:     .............
09:     void computeArea() {          <-- 추상 메소드를 오버라이딩하여 클래스 고유의 기능을 기술
10:         실제 원의 면적을 계산하는 기능이 기술됨;
11:     }
12: }
13:
14: public class Triangle extends Shape {
15:     .............
16:     void computeArea() {          <-- 추상 메소드를 오버라이딩하여 클래스 고유의 기능을 기술
17:         실제 삼각형의 면적을 계산하는 기능이 기술됨;
18:     }
19: }
```

Section 5.

추상 클래스와 다형성

5 추상 클래스와 다형성

- 추상 클래스와 추상 메소드의 오버라이딩을 이용하면 더욱 효율적인 다형성을 구현할 수 있다

5 추상 클래스와 다형성

예제 11.6 AbstractTest1.java

```
01: abstract class Shape {
02:     abstract void draw();
03:     abstract void computeArea(double a, double b);
04: }
05: class Rectangle1 extends Shape {
06:     void draw() {
07:         System.out.println("사각형을 그리는 기능");
08:     }
09:     void computeArea(double h, double v) {
10:         System.out.println("사각형의 넓이 : " + (h * v));
11:     }
12: }
13: class Triangle1 extends Shape {
14:     void draw() {
15:         System.out.println("삼각형을 그리는 기능");
16:     }
17:     void computeArea(double a, double h) {
18:         System.out.println("삼각형의 넓이 : " + (a * h / 2));
19:     }
20: }
21:
22: public class AbstractTest1 {
23:     public static void main(String args[]) {
24:         System.out.println("==추상 메소드를 이용한 다형성==");
25:         Shape s = new Rectangle1();
26:         s.draw();
27:         s.computeArea(5.0, 10.0);
28:         s = new Triangle1();
29:         s.draw();
30:         s.computeArea(5.0, 10.0);
31:     }
32: }
```

실행 결과

```
==추상 메소드를 이용한 다형성==
사각형을 그리는 기능
사각형의 넓이 : 50.0
삼각형을 그리는 기능
삼각형의 넓이 : 25.0
```

5 추상 클래스와 다형성

예제 11.7 AbstractTest2.java

```
01: abstract class Figure {                    하나의 추상 메소드를 가진 추상 클래스 생성
02:     abstract void draw();
03: }
04: class Triangle2 extends Figure {
05:     public void draw() {
06:         System.out.println("다형성 : 삼각형을 그린다");
07:     }
08: }
09: class Rectangle2 extends Figure {
10:     public void draw() {
11:         System.out.println("다형성 : 사각형을 그린다");
12:     }
13: }
14: class Oval2 extends Figure {
15:     public void draw() {
16:         System.out.println("다형성 : 타원형을 그린다");
17:     }
18: }
19: class Polydraw {  <-- 도형을 그리는 기능만을 위한 클래스 생성
20:     public void pdraw(Figure f) {
21:         f.draw();
22:     }
23:     /*
24:     public void pdraw(Triangle2 t) {
25:         t.draw();
26:     }
27:     public void pdraw(Rectangle2 r) {
28:         r.draw();
29:     }
30:     public void pdraw(Oval2 o) {
31:         o.draw();
32:     }
33:     */
34: }
35: public class AbstractTest2 {
36:     public static void main(String args[]) {
37:         Polydraw p = new Polydraw();
38:         Figure fg1 = new Triangle2();
39:         Figure fg2 = new Rectangle2();
40:         Figure fg3 = new Oval2();
41:         p.pdraw(fg1);
42:         p.pdraw(fg2);
43:         p.pdraw(fg3);
44:     }
45: }
```

완벽한 다형성을 제공하기 위해 pdraw() 메소드를 선언. 매개 변수로 Figure 형의 객체를 받아들여 그 객체의 draw() 메소드를 호출

다형성을 제공하지 않을 경우 각 객체별로 메소드를 작성해야 한다.

도형을 그리기 위해 Polydraw 클래스로부터 객체 생성

추상 클래스 형으로 객체 생성

동일한 객체에 동일한 메시지를 보낸다.

실행 결과

```
다형성 : 삼각형을 그린다
다형성 : 사각형을 그린다
다형성 : 타원형을 그린다
```

Section 6.

인터페이스

6 인터페이스

6-1 인터페이스 개요와 선언

- **인터페이스는 상수와 메소드 선언들만을 가진 클래스로 정의할 수 있다**
  - 추상 클래스와 유사하지만, 인터페이스는 앞에서 배운 추상 클래스보다 더욱 완벽한 추상화를 제공
  - 추상 클래스는 추상 메소드 외에 다른 멤버 변수나 일반 메소드를 가질 수 있지만, 인터페이스는 추상 메소드(메소드 선언만 있는)와 상수만을 가질 수 있다

- **인터페이스를 사용함으로써 완벽한 다중 상속은 아니지만 다중 상속을 흉내 낼 수 있다**

- **현재의 클래스가 이미 다른 클래스로부터 상속을 받고 있는 상태이면서, 또 다른 클래스의 요소들이 필요하다면 이때는 인터페이스를 사용하여야 한다**

6 인터페이스

6-1 인터페이스 개요와 선언

● 인터페이스의 형식과 예

[형식] interface

```
interface 인터페이스이름 [extends 인터페이스이름,[인터페이스이름],...] {
    .....상수 선언;
    .....메소드 선언;
}
```

예약어 interface로 선언. 인터페이스끼리 상속 관계를 구성할 수 있다.

[예]

```
01: interface Sleeper {
02:     public long ONE_SECOND = 1000;
03:     public long ONE_MINUTE = 60000;
04:     public void wakeup();
05: }
```

인터페이스의 선언

상수의 선언. 반드시 초기화해야 한다.

메소드의 선언. 끝에 ";"을 붙인다.

6 인터페이스

6-1 인터페이스 개요와 선언

- 인터페이스의 상속의 예

```
01: interface Sleeper {                                  ← 인터페이스 Sleeper 선언
02:     public long ONE_SECOND = 1000;
03:     public long ONE_MINUTE = 60000;
04:     public void wakeup();
05: }
06: interface Worker {                                   ← 인터페이스 Worker 선언
07:     public long WORK_TIME = 8;
08:     public void sleep();
09: }
10: interface People extends Sleeper, Worker {           ← 두 개의 인터페이스로부터 상속 받아
                                                           People 인터페이스 선언
11:     public int MAX = 24;
12:     public int MIN = 0;
13:     public void work();
14: }                                                    ← People 인터페이스를 포함하는 클래스 작성
15: public class InterfaceClass implements People {
16:     ...........
17:     public void wakeup() {                           ← 인터페이스 메소드 오버라이딩
18:         ...오버라이딩
19:     }
20:     public void sleep() {                            ← 인터페이스 메소드 오버라이딩
21:         ...오버라이딩
22:     }
23:     public void work() {                             ← 인터페이스 메소드 오버라이딩
24:         ...오버라이딩
25:     }
26: }
```

6 인터페이스

6-2 인터페이스의 사용

- **클래스에서 인터페이스를 사용하기 위해서는 implements 예약어 사용**
  - 클래스가 인터페이스를 포함하면, 인터페이스에서 선언된 모든 메소드를 오버라이딩 해야 한다

[형식] 인터페이스 포함(사용)

```
class 클래스 이름 extends 상위 클래스 이름 implements 인터 페이스 이름
                                        [,인터페이스 이름, .....] {

.... 멤버 변수 선언
.... 생성자
.... 메소드 선언
.... 인터페이스에 선언된 모든 메소드를 오버라이딩하여 구현하여야 한다.
}
```

6 인터페이스

6-2 인터페이스의 사용

```
01: interface Sleeper {  <-------------------- 인터페이스 Sleeper 선언
02:     public long ONE_SECOND = 1000;
03:     public long ONE_MINUTE = 60000;
04:     public void wakeup();
05: }
06: interface Worker {  <--------------------- 인터페이스 Worker 선언
07:     public long WORK_TIME = 8;
08:     public void sleep();
09: }
10: public class Man implements Sleeper, Worker {  <------ 두 개의 인터페이스를 포함
11:     public void wakeup() {  <--------------- 메소드 오버라이딩
12:         System.out.println("빨리 일어나 !!");
13:     }
14:     public void sleep() {  <---------------- 메소드 오버라이딩
15:         System.out.println("빨리 자 !!");
16:         ..........
17:     }
18: }
```

6 인터페이스

6-2 인터페이스의 사용

예제 11.8 InterfaceTest1.java

```
01: import java.util.Random;
02: interface IStack {                                  ← 인터페이스 선언. 두 개의 메소드 선언
03:     public void push(int item);
04:     public int pop();
05: }
06: class FixedStack implements IStack {                ← 인터페이스를 포함한 클래스 작성
07:     private int stack[];
08:     private int tos;
09:     FixedStack(int size) {
10:         stack = new int[size];
11:         tos = -1;
12:     }
13:     public void push(int item) {                    ← 인터페이스 메소드 오버라이딩
14:         if(tos==stack.length-1)
15:             System.out.println("스택이 꽉 찼음");
16:         else
17:             stack[++tos] = item;
18:     }
19:     public int pop() {                              ← 인터페이스 메소드 오버라이딩
20:         if(tos < 0) {
21:             System.out.println("스택이 비었음");
22:             return 0;
23:         }
24:         else
25:             return stack[tos--];
26:     }
27: }
```

```
28: public class InterfaceTest1 {
29:   public static void main(String args[]) {
30:       Random r = new Random();
31:       FixedStack mystack1 = new FixedStack(10);     ← 10개의 요소를 가지는 정수 스택 객체 생성
32:       for(int i=0 ; i<10 ; i++)
33:           mystack1.push(r.nextInt(10));             ← 난수를 발생시켜 스택에 저장
34:       System.out.println("스택 : mystack1");
35:       for(int i=0 ; i<10 ; i++)
36:           System.out.print(mystack1.pop() + "  ");  ← 스택의 내용을 출력
37:   }
38: }
```

실행 결과

```
스택 : mystack1
9 5 3 3 4 4 1 0 0 8
```

Section 7.

인터페이스와 다형성

7 인터페이스와 다형성

- **인터페이스도 추상 클래스와 같이 다형성을 구현하는 데 사용될 수 있다**
  - 추상 클래스를 이용하여 다형성을 구현하는 것과 동일한 형태

7 인터페이스와 다형성

예제 11.9 InterfaceTest2.java

```
01: abstract class Figure1 {
02:     abstract void draw();
03: }
04: interface Shape1 {
05:     public void computeArea(double a, double b);
06: }
07: class Triangle3 extends Figure1 implements Shape1 {
08:     void draw() {
09:         System.out.println("삼각형을 그리는 기능");
10:     }
11:    public void computeArea(double a, double h) {
12:         System.out.println("삼각형의 넓이 : " + (a * h / 2));
13:     }
14: }
15: class Rectangle3 extends Figure1 implements Shape1 {
16:     void draw() {
17:         System.out.println("사각형을 그리는 기능");
18:     }
19:     public void computeArea(double h, double v) {
20:         System.out.println("사각형의 넓이 : " + (h * v));
21:     }
22: }
```

추상 클래스 선언

인터페이스 선언

추상 클래스와 인터페이스를 포함하는 클래스 작성

메소드 오버라이딩

메소드 오버라이딩

7 인터페이스와 다형성

```
23: class Oval3 extends Figure1 implements Shape1 {
24:     void draw() {
25:         System.out.println("원을 그리는 기능");
26:     }
27:     public void computeArea(double r1, double r2) {
28:         System.out.println("원의 넓이 : " + (3.14 * r1 * r2));
29:     }
30: }
31: class Polydraw1 {
32:     public void pdraw(Figure1 f) {
33:         f.draw();
34:     }
35:     public void pcomputeArea(Shape1 s,double a, double b) {
36:         s.computeArea(a,b);
37:     }
38: }
39: public class InterfaceTest2 {
40:     public static void main(String args[]) {
41:         Polydraw1 p = new Polydraw1();
42:         Figure1 fg1 = new Triangle3();
43:         Figure1 fg2 = new Rectangle3();
44:         Figure1 fg3 = new Oval3();
45:         Shape1 sp1 = new Triangle3();
46:         Shape1 sp2 = new Rectangle3();
47:         Shape1 sp3 = new Oval3();
48:         p.pdraw(fg1);
49:         p.pcomputeArea(sp1,4,4);
50:         p.pdraw(fg2);
51:         p.pcomputeArea(sp2,4,4);
52:         p.pdraw(fg3);
53:         p.pcomputeArea(sp3,4,4);
54:     }
55: }
```

인터페이스 형의 객체 변수에 클래스의 객체를 배정

인터페이스 형의 객체 변수로 메소드 호출

인터페이스 형의 객체 변수로 메소드 호출

인터페이스 형의 객체 변수로 메소드 호출

실행 결과

```
삼각형을 그리는 기능
삼각형의 넓이 : 8.0
사각형을 그리는 기능
사각형의 넓이 : 16.0
원을 그리는 기능
원의 넓이 : 50.24
```

Thank You!

Copyright© 2023 Hanbit Academy, Inc.
All rights reserved.