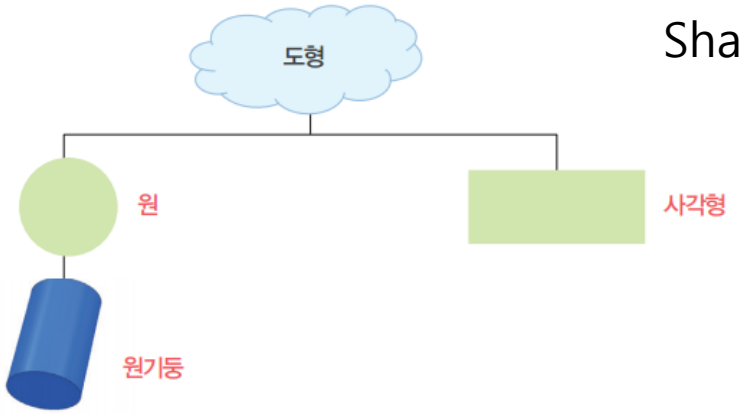
Chapter 07

추상 클래스와 인터페이스

Contents

추상 클래스의 개념



Shape 클래스의 draw()?

- 추상 메서드
  - 메서드 본체를 완성하지 못한 메서드. 무엇을 할지는 선언할 수 있지만, 어떻게 할지는 정의할 수 없음

- 추상 클래스
  - 보통 하나 이상의 추상 메서드를 포함하지만 없을 수도 있음
  - 주로 상속 계층에서 자식 멤버의 이름을 통일하기 위하여 사용

```
추상클래스 s = new 추상클래스();   // 추상 클래스는 인스턴스를 생성하지 못한다.
```

추상 클래스의 선언

- 추상 클래스 선언

```
abstract class 클래스이름 {
    // 필드
    // 생성자
    // 메서드
}
```

추상 클래스라는 것을 나타낸다.

일반적으로 하나 이상의 추상 메서드를 포함한다.

- 추상 메서드 선언

```
abstract 반환타입 메서드이름() ;
```

추상 메서드라는 것을 나타낸다.

메서드 본체가 없다.

항상 세미콜론으로 끝나야 한다.

- 예제 : [예제 7-1]

```
3  abstract class Shape {
4      double pi = 3.14;
5
6      abstract void draw();
7
8      public double findArea() {
9          return 0.0;
10     }
11 }
```

- 예제 : [예제 7-2]

```
3  class Circle extends Shape {
4      int radius;
5
6      public Circle(int radius) {
7          this.radius = radius;
8      }
9
10     public void draw() {
11         System.out.println("원을 그리다.");
12     }
13
14     public double findArea() {
15         return pi * radius * radius;
16     }
17 }
```

```
3  class Rectangle extends Shape {
4      int width, height;
5
6      public Rectangle(int width, int height) {
7          this.width = width;
8          this.height = height;
9      }
10
11     public void draw() {
12         System.out.println("사각형을 그리다.");
13     }
14
15     public double findArea() {
16         return width * height;
17     }
18 }
```

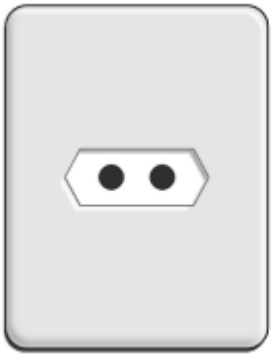
- 예제 : [예제 7-3]

```
3  public class AbstractClassDemo {
4      public static void main(String[] args) {
5          // Shape s = new Shape();
6
7          Circle c = new Circle(3);
8          c.draw();
9          System.out.printf("원의 넓이는 %.1f\n", c.findArea());
10
11         Rectangle r = new Rectangle(3, 4);
12         r.draw();
13         System.out.printf("사각형의 넓이는 %.1f\n", r.findArea());
14     }
15 }
```

```
원을 그리다.
원의 넓이는 28.3
사각형을 그리다.
사각형의 넓이는 12.0
```

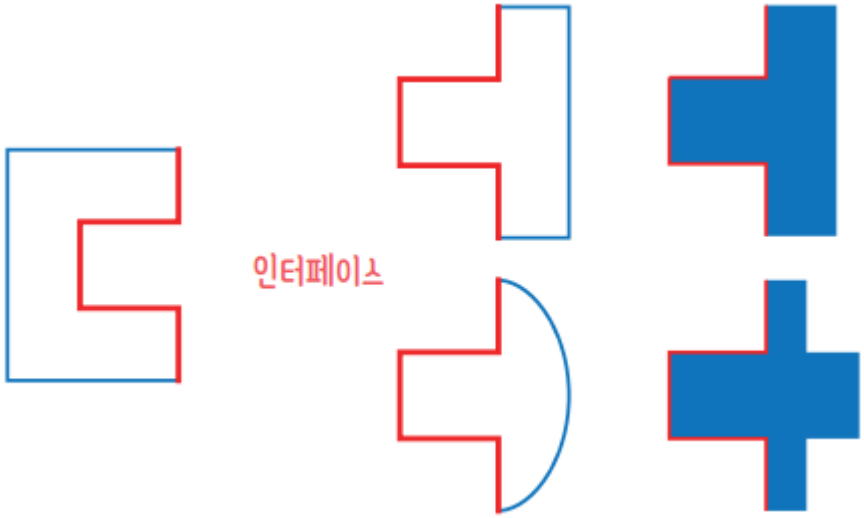
인터페이스 개요(1)

인터페이스만
맞으면
건축한 후에
어떤 가전제품들이
들어올지 신경 쓸
필요 없다.



인터페이스만
맞으면
어떤 가전제품도
사용할 수 있다.

현실 세계의 인터페이스



자바의 인터페이스

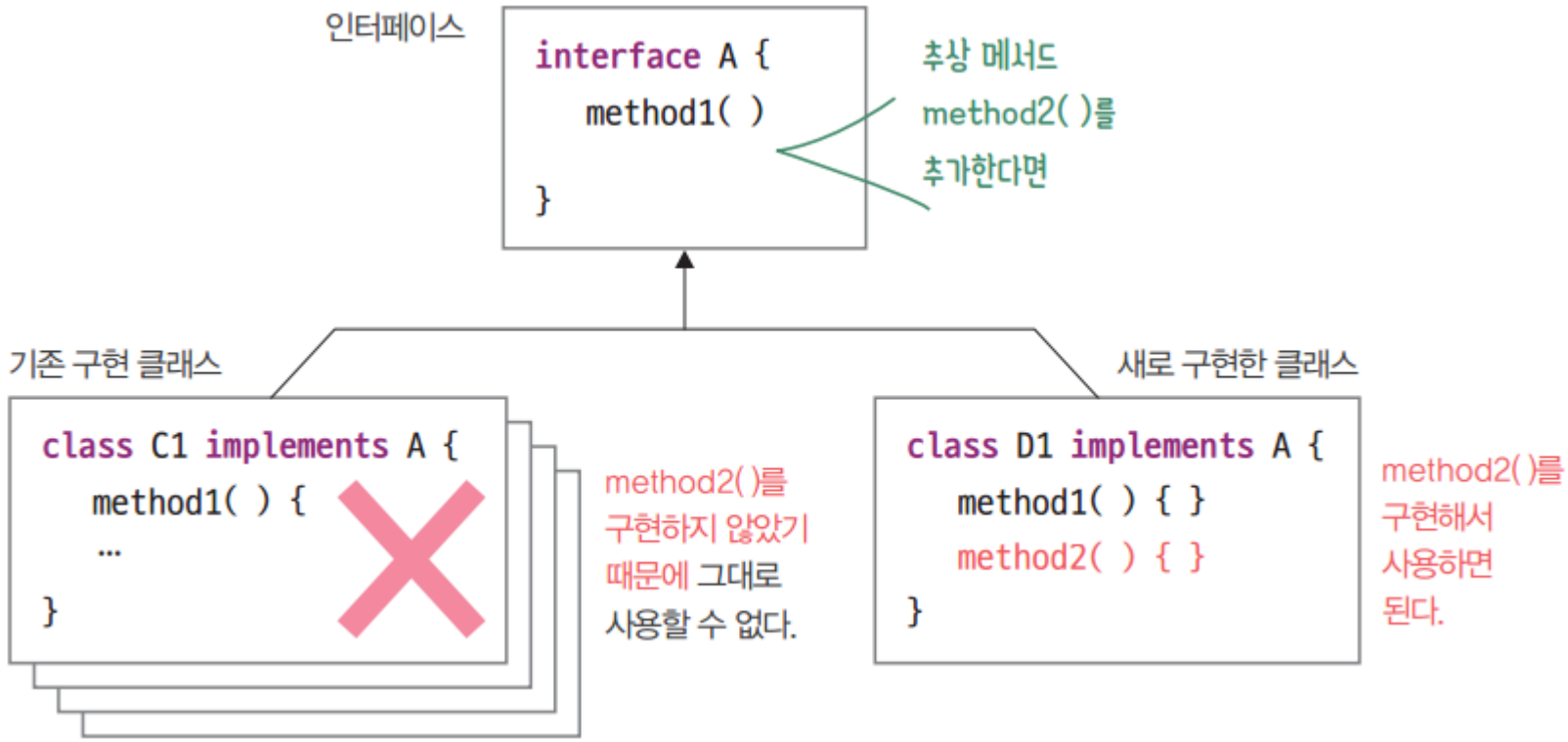
인터페이스 개요(2)

- 인터페이스에 의한 장점
  - 인터페이스만 준수하면 통합에 신경 쓰지 않고 다양한 형태로 새로운 클래스를 개발할 수 있다.
  - 클래스의 다중 상속을 지원하지 않지만, 인터페이스로 다중 상속 효과를 간접적으로 얻을 수 있다.
- 인터페이스와 추상 클래스의 차이

분류	인터페이스	추상 클래스
구현 메서드	포함 불가(단, 디폴트 메서드와 정적 메서드는 예외)	포함 가능
인스턴스 변수	포함 불가능	포함 가능
다중 상속	가능	불가능
디폴트 메서드	선언 가능	선언 불가능
생성자와 main( )	선언 불가능	선언 가능
상속에서의 부모	인터페이스	인터페이스, 추상 클래스
접근 범위	모든 멤버를 공개	추상 메서드를 최소한 자식에게 공개

디폴트 메서드와 정적 메서드(1)

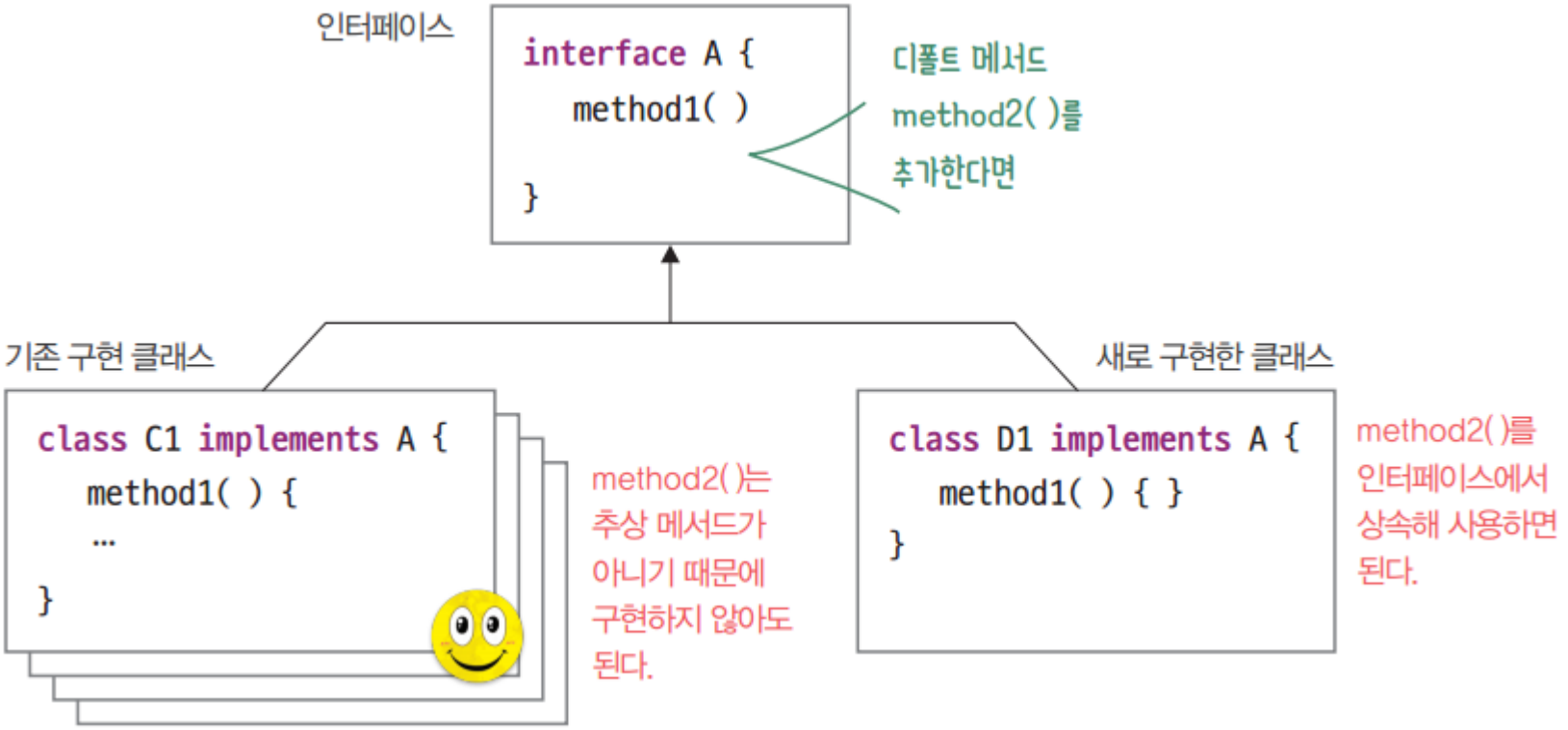
- 인터페이스 수정이 기존 구현 클래스에 미치는 영향



디폴트 메서드와 정적 메서드(2)

- 디폴트 메서드

```
default 반환타입 디폴트메서드이름( ) {
   // 본체를 구성하는 코드
}
```



디폴트 메서드와 정적 메서드(3)

- 디폴트 메서드는 오버라이딩될 수 있지만, 정적 메서드는 오버라이딩될 수 없다.
- 디폴트 메서드는 인스턴스 메서드이므로 객체를 생성한 후 호출하지만, 정적 메서드는 인터페이스로 직접 호출한다.

인터페이스의 구조

- 인터페이스의 구조

```
interface 인터페이스이름 {
    // 상수 필드
    // 추상 메서드
    // 디폴트 메서드
    // 정적 메서드
}
```

- 상수 필드 : 상수만 가능하기 때문에 public static final 키워드가 없어도 된다.
- 추상 메서드 : interface의 모든 메서드가 public abstract이기 때문에 public abstract를 생략해도 된다.
- 디폴트 메서드 : JDK 8부터 선언할 수 있다.

- 인터페이스 파일의 컴파일

```
interface MyInterface {
    int MAX = 10;

    void sayHello();
}
```

MyInterface.java

컴파일

```
interface MyInterface {
    public static final int MAX = 10;

    public abstract void sayHello();
}
```

MyInterface.class

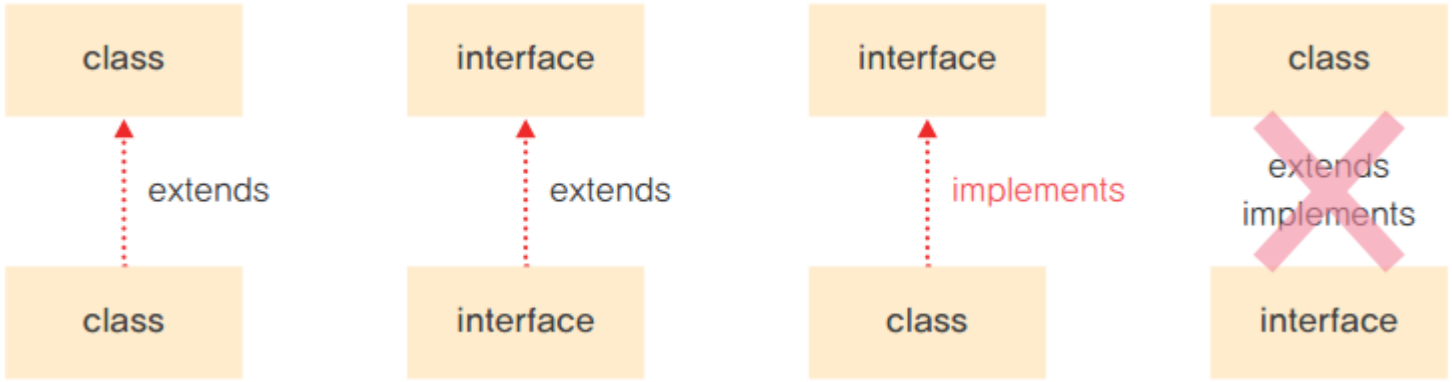
인터페이스의 상속(1)

- 인터페이스 상속

```
// 인터페이스를 상속하려면 extends 키워드를 사용한다.
interface 자식인터페이스 extends 부모인터페이스 {
}

// 인터페이스를 구현하려면 implements 키워드를 사용한다.
class 자식클래스 implements 부모인터페이스 {
}
```

- 클래스와 인터페이스의 관계



인터페이스의 상속(2)

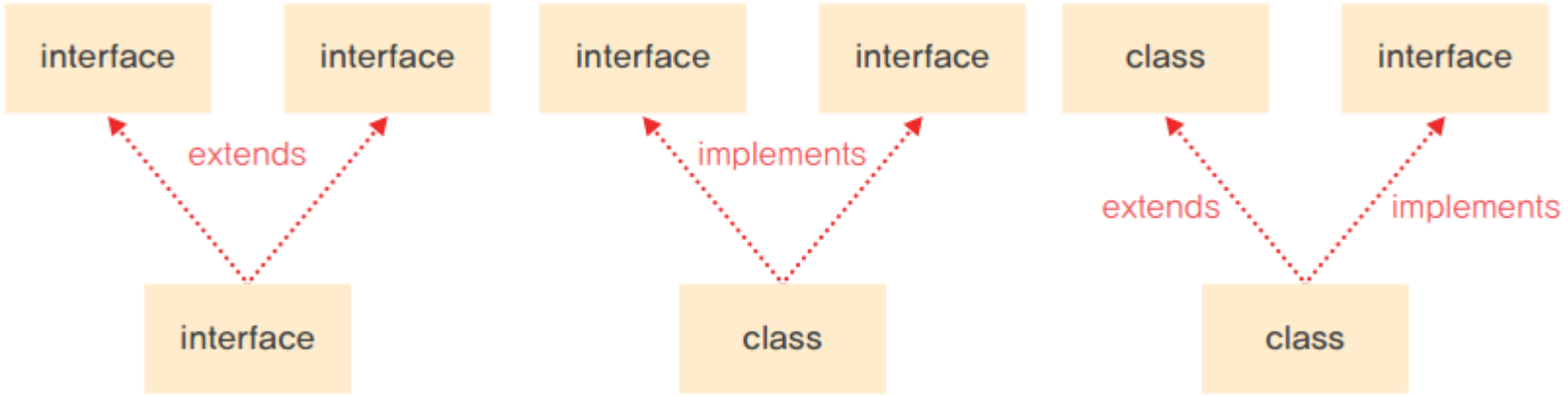
```
// 상속할 인터페이스가 여러 개라면 쉼표(,)로 연결한다.
interface 자식인터페이스 extends 부모인터페이스1, 부모인터페이스2 {
}
class 자식클래스 implements 부모인터페이스1, 부모인터페이스2 {
}
```

```
// 인터페이스는 다중 상속할 수 있다.
class 자식클래스 extends 부모클래스 implements 부모인터페이스1, 부모인터페이스2 {
}

// 클래스는 다중 상속할 수 없다.
class 자식클래스 extends 부모클래스1, 부모클래스2 {
}
```

인터페이스의 상속(3)

- 인터페이스를 이용한 다중 상속 효과



인터페이스와 상수(1)

- 예제 : [예제 7-4]

[예제 7-4] 상수를 정의한 인터페이스 sec03/Coin1Demo.java

```
01  interface Coin {
02      int PENNY = 1, NICKEL = 5, DIME = 10, QUARTER = 25;
03  }
04
05  public class Coin1Demo {
06      public static void main(String[] args) {
07          System.out.println("Dime은 " + Coin.DIME + "센트입니다.");
08      }
09  }
```

int만 표시되어 있지만, public static final int이다. 인터페이스의 모든 필드는 public static final이기 때문이다.

인터페이스에 정의된 상수는 DIME을 의미한다.

```
Dime은 10센트입니다.
```

인터페이스와 상수(2)

- 예제 : [예제 7-5]

[예제 7-5] 구현 클래스로 테스트 sec03/Coin2Demo.java

```
01  public class Coin2Demo implements Coin {
02      public static void main(String[] args) {
03
04          System.out.println("Dime은 " + DIME + "센트입니다.");
05
06      }
07  }
```

Coin 인터페이스를 구현한다. Coin 인터페이스가 추상 메서드를 포함하지 않으므로 추가할 코드가 없다.

Coin 인터페이스의 구현 클래스이므로 직접 상수를 사용할 수 있다.

Comparable 인터페이스

- 정의

```
public interface Comparable {
  int compareTo(Object other);
}
```

다른 객체보다 크면 양수, 동일하면 0, 작으면 음수를 반환한다.

- 예제 : [예제 7-6]

```
3  class Circle implements Comparable {
4      double radius;
5
6      public Circle(double radius) {
7          this.radius = radius;
8      }
9
10     public int compareTo(Object o) {
11         Circle c = (Circle) o;
12         if (this.radius > c.radius)
13             return 1;
14         else if (this.radius == c.radius)
15             return 0;
16         else
17             return -1;
18     }
19 }
```

```
21 public class CircleDemo {
22     public static void main(String[] args) {
23         Circle c1 = new Circle(5.0);
24         Circle c2 = new Circle(6.0);
25
26         if (c1.compareTo(c2) > 0)
27             System.out.println("첫 번째 원이 두 번째 원보다 크다.");
28         else if (c1.compareTo(c2) == 0)
29             System.out.println("두 원의 크기가 같다.");
30         else
31             System.out.println("첫 번째 원이 두 번째 원보다 작다.");
32     }
33 }
```

인터페이스의 상속과 구현 클래스

- 예 : 전자제품에 포함되어야 하는 제어부가 요구하는 조건
  - 모든 전자제품에는 전원을 온·오프하는 기능이 있으며, 수리 및 공장 초기화를 할 수 있다.
  - 전자제품 객체는 turnOn( ) 메서드, turnOff( ) 메서드로만 전원을 조절할 수 있어야 한다.
  - 수리 및 공장 초기화 기능을 미리 구현해 놓아서 필요할 때 사용할 수 있어야 한다.
  - 수리 기능은 자식 클래스에서 오버라이딩할 수도 있다.

- 예제 : [예제 7-7]

```
3  public interface Controllable {
4      default void repair() {
5          System.out.println("장비를 수리한다.");
6      }
7
8      static void reset() {
9          System.out.println("장비를 초기화한다.");
10     }
11
12     void turnOn();
13     void turnOff();
14 }
```

- 예제 : [예제 7-8]

```
3  public interface RemoteControllable extends Controllable {
4      void remoteOn();
5
6      void remoteOff();
7  }
```

- 예제 : [예제 7-9]

```java
public class TV implements Controllable {

    @Override
    public void turnOn() {
        System.out.println("TV를 켠다.");
    }

    @Override
    public void turnOff() {
        System.out.println("TV를 끈다.");
    }
}
```

```java
public class Computer implements Controllable {
    public void turnOn() {
        System.out.println("컴퓨터를 켠다.");
    }

    public void turnOff() {
        System.out.println("컴퓨터를 끈다.");
    }
}
```

- 예제 : [예제 7-10]

```
3  public class ControllableDemo {
4      public static void main(String[] args) {
5          TV tv = new TV();
6          Computer com = new Computer();
7
8          tv.turnOn();
9          tv.turnOff();
10         tv.repair();
11         com.turnOn();
12         com.turnOff();
13         com.repair();
14         Controllable.reset();
15         // tv.reset();
16         // com.reset();
17     }
18 }
```

```
TV를 켠다.
TV를 끈다.
장비를 수리한다.
컴퓨터를 켠다.
컴퓨터를 끈다.
장비를 수리한다.
장비를 초기화한다.
```

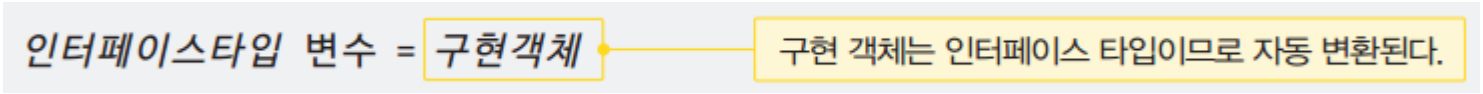
- 예제 : [예제 7-11]

```
3  interface Portable {
4      void inMyBag();
5  }
6
7  public class Notebook extends Computer implements Portable {
8      public void inMyBag() {
9          System.out.println("노트북은 가방에 있다.");
10     }
11
12     public void turnOn() {
13         System.out.println("노트북을 켠다.");
14     }
15
16     public void turnOff() {
17         System.out.println("노트북을 끈다.");
18     }
19
20     public static void main(String[] args) {
21         Notebook n = new Notebook();
22
23         n.turnOn();
24         n.turnOff();
25         n.inMyBag();
26     }
27 }
```

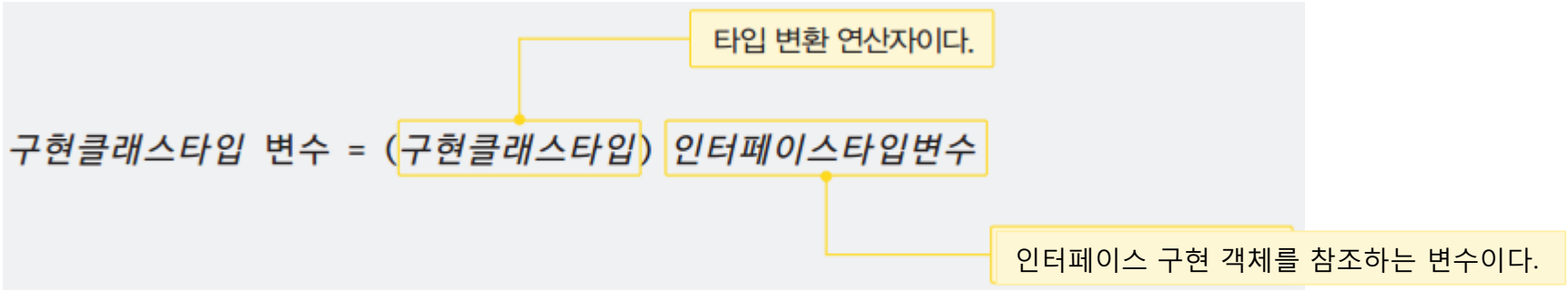
```
노트북을 켠다.
노트북을 끈다.
노트북은 가방에 있다.
```

인터페이스 타입

- 인터페이스도 클래스처럼 하나의 타입이므로 변수를 인터페이스 타입으로 선언 가능
- 인터페이스의 구현 클래스는 그 인터페이스의 자식 타입



- 인터페이스 타입 변수가 구현 객체를 참조한다면 강제 타입 변환 가능



타입 변환과 다형성

```
interface Movable {
    void move();
}
```

```
class Car implements Movable {
    public void move() { … }
    public void show() { … }
}
```

Movable 타입에는
move( ) 메서드만 있고
show( ) 메서드는 없다.

Car 타입에는
move( ), show( )
메서드 둘 다 있다.

MovableDemo.java

```
Movable m = new Car();
m.move();

// m.show();
Car c = (Car) m; // 강제 타입 변환

c.move();
c.show()
```

- 예제 : [예제 7-12]

```
7  public class ControllableDemo {
8      public static void main(String[] args) {
9          Controllable[] controllable = { new TV(), new Computer() };
10
11         for (Controllable c : controllable) {
12             c.turnOn();
13             c.turnOff();
14             c.repair();
15         }
16         Controllable.reset();
17     }
18 }
```

```
TV를 켠다.
TV를 끈다.
장비를 수리한다.
컴퓨터를 켠다.
컴퓨터를 끈다.
장비를 수리한다.
장비를 초기화한다.
```

- 예제 : [예제 7-13]

```java
interface Animal {
    void sound();
}

class Dog implements Animal {
    public void sound() {
        System.out.println("멍멍~~");
    }
}

class Cuckoo implements Animal {
    public void sound() {
        System.out.println("뻐꾹~~");
    }
}

public class AnimalDemo {
    public static void main(String[] args) {
        Dog d = new Dog();
        Cuckoo c = new Cuckoo();

        makeSound(d);
        makeSound(c);
    }

    static void makeSound(Animal a) {
        a.sound();
    }
}
```

```
멍멍~~
뻐꾹~~
```

- 예제 : [예제 7-14]

```java
interface Movable {
    void move(int x);
}

class Car implements Movable {
    private int pos = 0;

    public void move(int x) {
        pos += x;
    }

    public void show() {
        System.out.println(pos + "m 이동했습니다.");
    }
}

public class MovableDemo {
    public static void main(String[] args) {
        Movable m = new Car();

        m.move(5);
        // m.show();

        Car c = (Car) m;
        c.move(10);
        c.show();
    }
}
```

```
15m 이동했습니다.
```