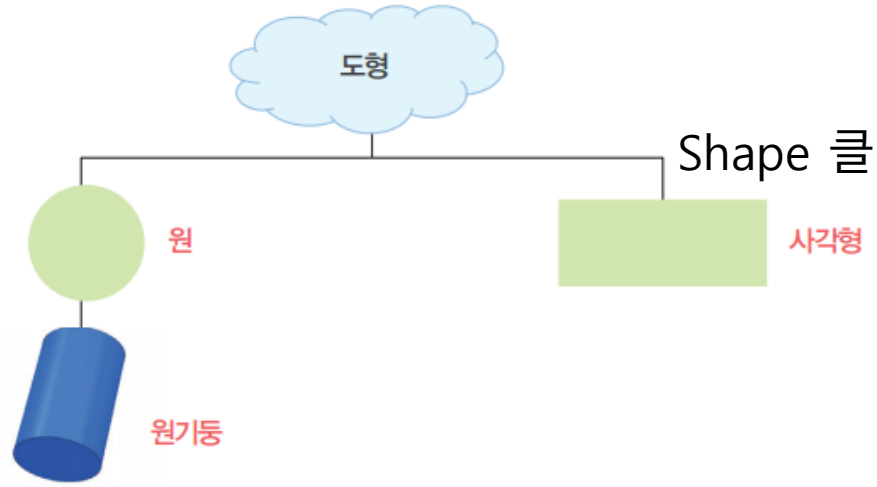
추상클래스와 인터페이스

백석대학교 강윤희

추상 클래스



Shape 클래스의 draw()?

- **추상 메서드**
  - 메서드 본체를 완성하지 못한 메서드
  - 무엇을 할지는 선언할 수 있지만, 어떻게 할지는 정의할 수 없음 (내용이 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 클래스

추상 클래스 선언

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

- `abstract`: 추상 클래스라는 것을 나타낸다.
- `// 메서드`: 일반적으로 하나 이상의 추상 메서드를 포함한다.

추상 메서드 선언

```
abstract 반환타입 메서드이름();
```

- `abstract`: 추상 메서드라는 것을 나타낸다.
- `;`: 항상 세미콜론으로 끝나야 한다.
- `()` 뒤 `;`: 메서드 본체가 없다.

예제

- sec01/Shape.java,
- sec01/Circle.java,
- sec01/AbstractClassDemo.java

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

추상 클래스

```java
abstract class Shape {
    double pi = 3.14;

    abstract void draw();

    public double findArea() {
        return 0.0;
    }
}
```

```java
class Circle extends Shape {
    int radius;

    public Circle(int radius) {
        this.radius = radius;
    }

    public void draw() {
        System.out.println("Draw circle");
    }

    public double findArea() {
        return pi * radius * radius;
    }
}
```

```java
class Rectangle extends Shape {
    int width, height;

    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }

    public void draw() {
        System.out.println("Draw Rectangle");
    }

    public double findArea() {
        return width * height;
    }
}
```

추상 클래스

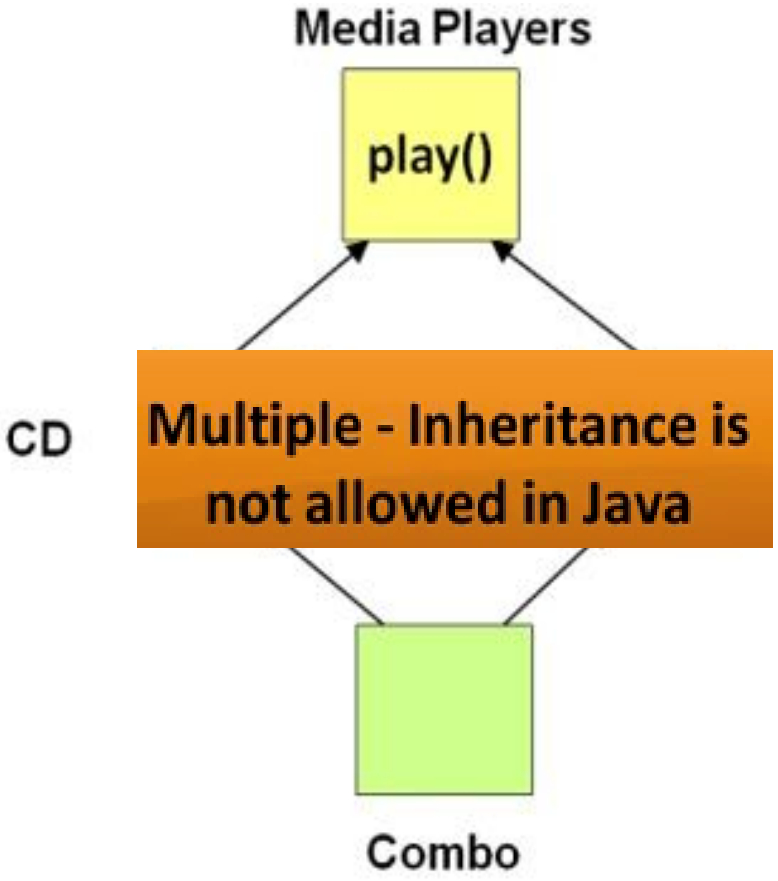
```java
public class AbstractClassDemo {
    public static void main(String[] args) {
        // Shape s = new Shape();

        Circle c = new Circle(3);
        c.draw();
        System.out.printf("Area of Circle %.1f\n", c.findArea());

        Rectangle r = new Rectangle(3, 4);
        r.draw();
        System.out.printf("Area of Rectangle %.1f\n", r.findArea());
    }
}
```

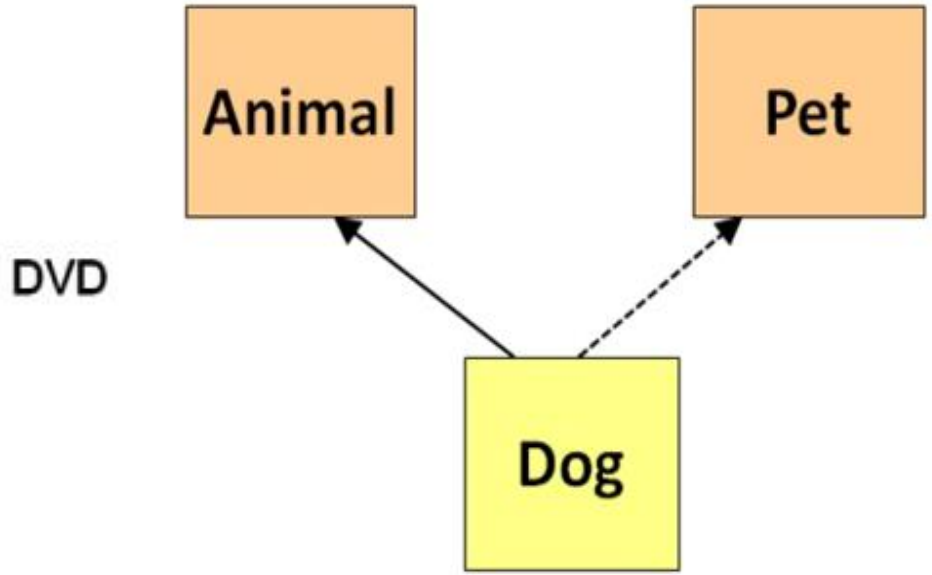
```
<terminated> AbstractClassDemo
Draw circle
Area of Circle 28.3
Draw Rectangle
Area of Rectangle 12.0
```

Abstract class ≈ Interface (비슷하지만 같지 않음)

- **Similarities**
  - Abstract methods
- **Differences**
  - 1 vs. Many(100%)
  - No inheritance

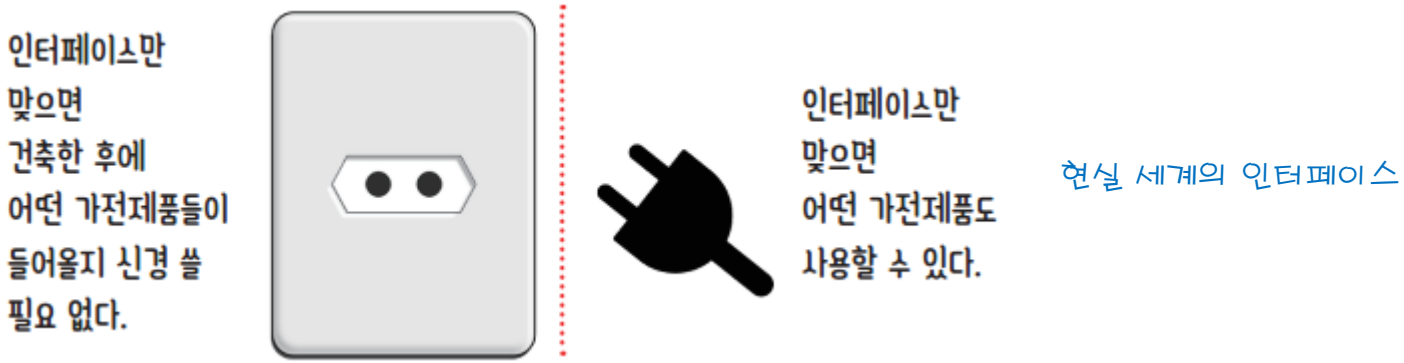
Why Interface is required in Java



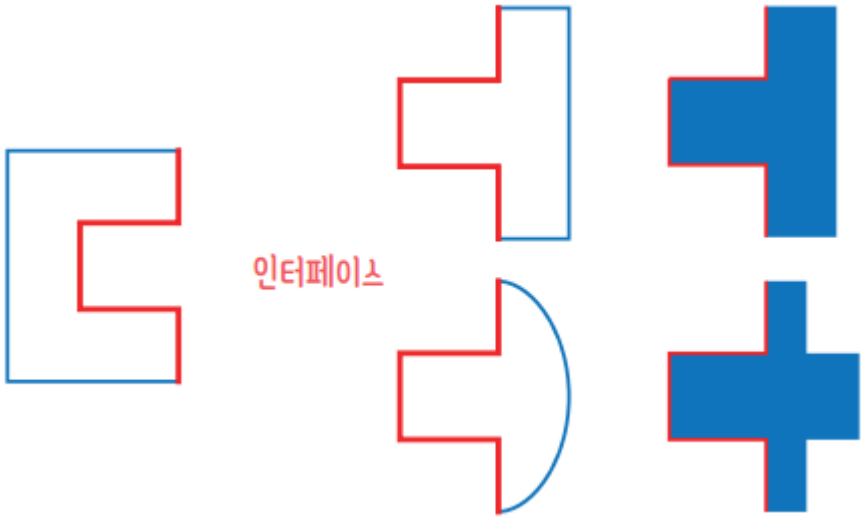


인터페이스 개념

■ 현실 세계와 자바의 인터페이스



현실 세계의 인터페이스



자바의 인터페이스

인터페이스의 선언

- 자바 인터페이스
  - 하나 이상의 **추상 메소드**로 선언됨
  - 추상 메소드는 서브 클래스에 의해 반드시 구현되어야 함
  - 인터페이스도 클래스와 같이 **상속**될 수 있음

- 인터페이스의 선언 및 사용 예

```
public interface <인터페이스이름> [extends <인터페이스이름1>, ...]
{
        ..... // 상수 및 메소드 선언
 }

public interface CalculatorInterface {    // 사용 예 : 계산기 인터페이스
      public abstract int add(int x, int y);
          ...
}
```

인터페이스 개념

인터페이스에 의한 장점

- 인터페이스만 준수하면 통합에 신경 쓰지 않고 다양한 형태로 새로운 클래스를 개발할 수 있음
- 클래스의 다중 상속을 지원하지 않지만, 인터페이스로 다중 상속효과를 얻을 수 있음

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

인터페이스의 구현

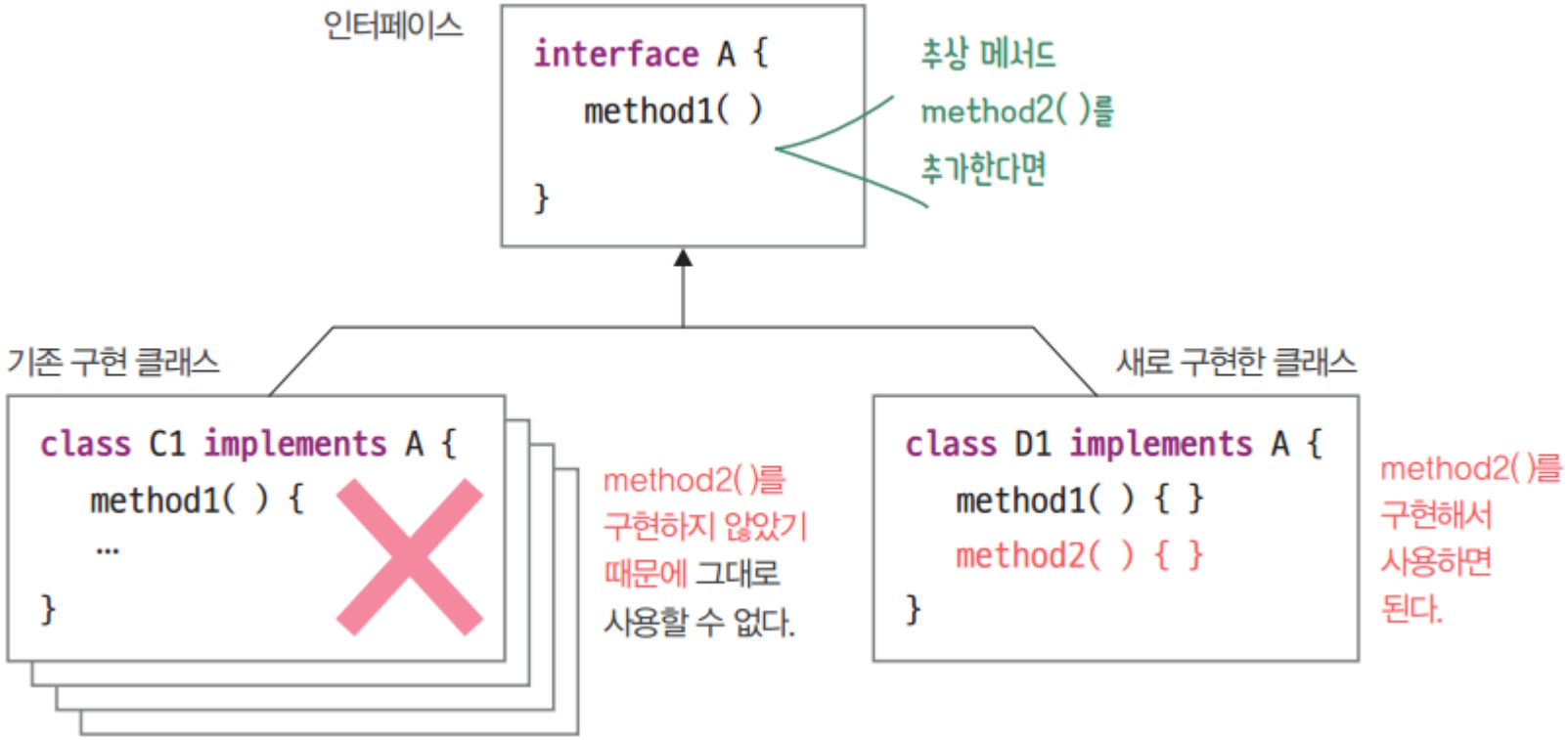
- 인터페이스는 이 인터페이스를 제공하고자 하는 클래스에 의해 구현됨
  - 인터페이스를 구현하는 클래스는 "implements" 예약어를 사용
  - 인터페이스에 선언된 모든 메소드는 오버라이딩되어 구현되어야 함

- 인터페이스를 구현하는 클래스의 선언형식과 사용 예

```
[public/final/abstract] class <클래스이름> extends <부모클래스이름>
implements <인터페이스이름1> [, <인터페이스이름2>, .....] {
        .... // 멤버변수선언
        .... // 생성자
        .... // 메소드선언, 인터페이스의 모든 메소드를 오버라이딩
}

public calculator implements CalculatorInterface { // 사용 예
      public int add(int x, int y) {
           return(x + y);          // x와 y를 더하여 반환
      }
}
```

인터페이스 기초

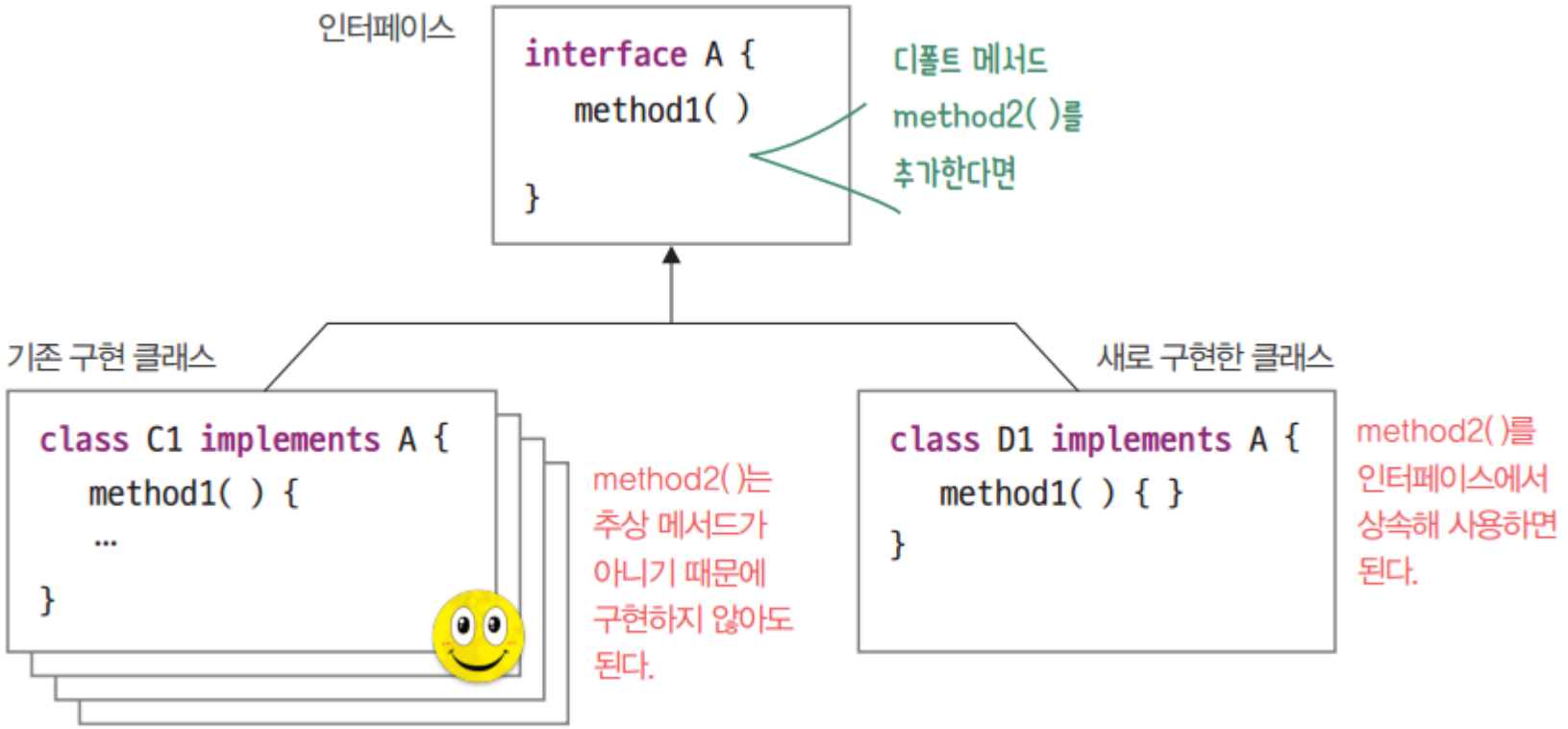
- 인터페이스 수정이 기존 구현 클래스에 미치는 영향



인터페이스 기초

■ 디폴트 메서드

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



인터페이스 기초

디폴트 메서드와 정적 메서드

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

인터페이스 구조

```
interface 인터페이스이름 {
    // 상수 필드              → 상수만 가능하기 때문에 public static final 키워드 생략 가능

    // abstract 메서드        → 인터페이스의 모든 메서드(아래 3가지 종류를 제외)가
                                public abstract이기 때문에 public abstract 키워드 생략 가능

    // default 메서드         → JDK 8부터 가능

    // static 메서드          → JDK 8부터 가능

    // private 메서드         → JDK 9부터 가능
}
```

인터페이스 기초

인터페이스 파일의 컴파일

```
interface MyInterface {
   int MAX = 10;

   void sayHello();
}
```

MyInterface.java


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

   public abstract void sayHello();
}
```

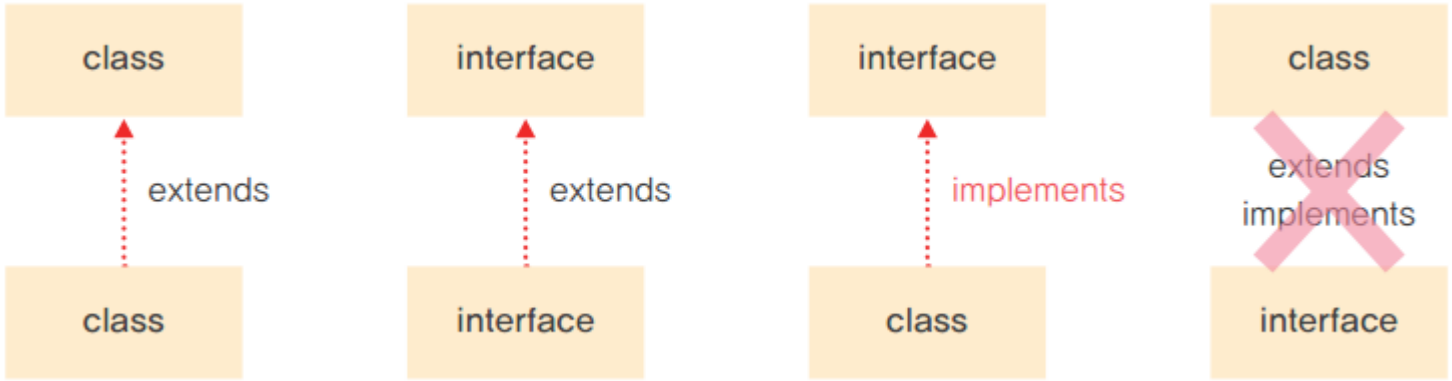
MyInterface.class

인터페이스 기초

인터페이스 상속

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

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

클래스와 인터페이스의 관계



인터페이스 기초

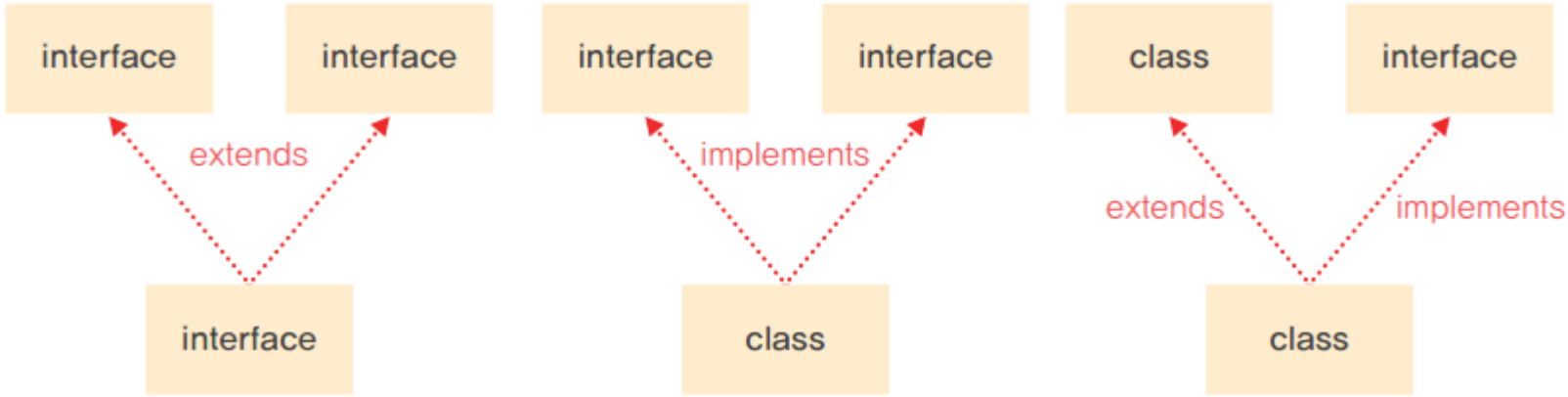
인터페이스 상속

```
// 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 {
}
```

인터페이스 기초

- 인터페이스를 이용한 다중 상속 효과



인터페이스 응용

인터페이스와 상수

```java
interface Coin {
    int PENNY = 1, NICKEL = 5, DIME = 10, QUARTER = 25;
}

public class Coin1Demo {
    public static void main(String[] args) {
        System.out.println("Dime is " + Coin.DIME + " cent");
    }
}
```

```
<terminated> Coin1Dem
Dime is 10cent
```

```java
public class Coin2Demo implements Coin {
    public static void main(String[] args) {
        System.out.println("Dime is " + DIME + " cent");
    }
}
```

```
<terminated> Coin2Demo [Jav
Dime is 10 cent
```

인터페이스 응용

- **Comparable 인터페이스**
  - 정의

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

다른 객체보다 크면 양수, 동일하면 0,
작으면 음수를 반환한다.

- **Circle 클래스에서 원의 비교를 위해 사용**

인터페이스 응용

```
1  class Circle implements Comparable {
2      double radius;
3
4      public Circle(double radius) {
5          this.radius = radius;
6      }
7
8      public int compareTo(Object o) {
9          Circle c = (Circle) o;
10         if (this.radius > c.radius)
11             return 1;
12         else if (this.radius == c.radius)
13             return 0;
14         else
15             return -1;
16     }
17 }
18
19 public class CircleDemo {
20     public static void main(String[] args) {
21         Circle c1 = new Circle(5.0);
22         Circle c2 = new Circle(6.0);
23
24         if (c1.compareTo(c2) > 0)
25             System.out.println("First circle (c1) is larger than second one(c2)");
26         else if (c1.compareTo(c2) == 0)
27             System.out.println("Two circles are equal");
28         else
29             System.out.println("First circle (c1) is smaller than second one(c2)");
30     }
31 }
```

인터페이스 응용

인터페이스의 상속과 구현 클래스

- 전자제품에 포함되어야 하는 제어부의 요구 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.

```
1  public interface Controllable {
2      default void repair() {
3          System.out.println("Repair a device");
4      }
5
6      static void reset() {
7          System.out.println("Reset a device");
8      }
9
10     void turnOn();
11     void turnOff();
12 }
```

인터페이스 응용

인터페이스의 상속과 구현 클래스

- RemoteControllable 는 4 + 2 메소드를 가짐

```
1
2  public interface RemoteControllable extends Controllable {
3      void remoteOn();
4
5      void remoteOff();
6  }
```

```
1  public interface Controllable {
2      default void repair() {
3          System.out.println("Repair a device");
4      }
5
6      static void reset() {
7          System.out.println("Reset a device");
8      }
9
10     void turnOn();
11     void turnOff();
12 }
```

인터페이스 응용

- 인터페이스의 상속과 **구현 클래스**
  - 모든 인터페이스를 구현해야 함

```java
public class TV implements Controllable {

    @Override
    public void turnOn() {
        System.out.println("Turn on TV");
    }

    @Override
    public void turnOff() {
        System.out.println("Turn off TV");
    }
}
```

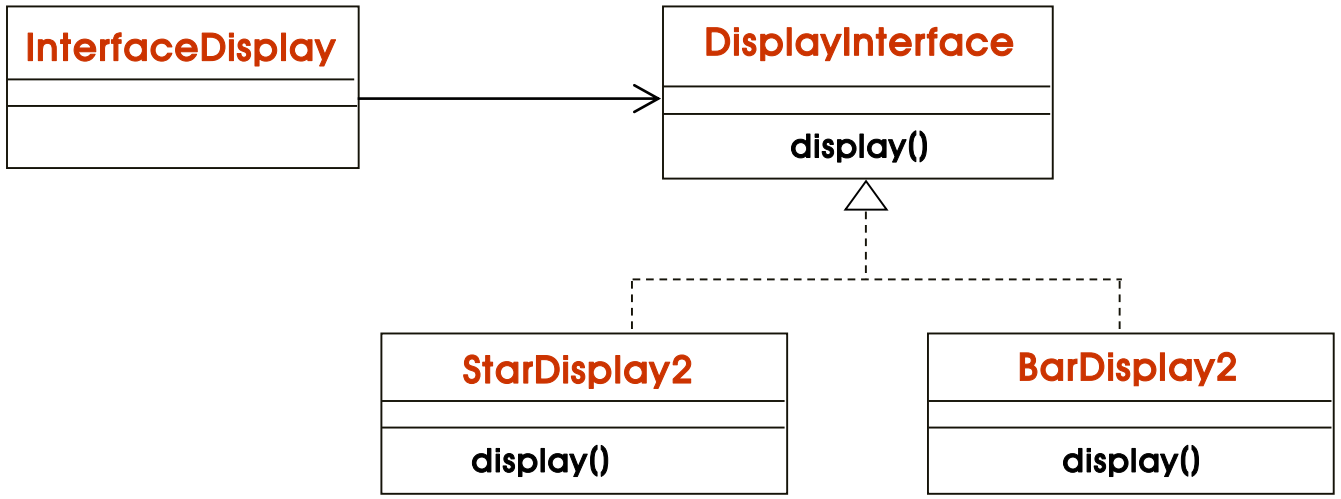
```java
public interface Controllable {
    default void repair() {
        System.out.println("Repair a device");
    }

    static void reset() {
        System.out.println("Reset a device");
    }

    void turnOn();
    void turnOff();
}
```

실습문제 – 기초

인터페이스를 이용한 문자열 출력 : InterfaceDisplay.java

- 문자열 출력을 위한 추상 메소드를 선언하는 인터페이스와 이 인터페이스를 구현한 두 개의 클래스를 이용하여 문자열 출력을 실습한다.
  - DisplayInterface 인터페이스 : 추상메소드 display()를 선언한다.
  - StarDisplay2 클래스 : 문자열의 좌우측에 별표(*)를 출력하도록 display()를 구현한다.
  - BarDisplay2 클래스 : 문자열의 좌우측에 바(–)를 출력하도록 display()를 구현한다.

- InterfaceDisplay 클래스는 StarDisplay2와 BarDisplay2의 객체를 각각 생성하고 문자열 출력을 의뢰한다.



(실습)인터페이스 구현

```
interface DisplayInterface
{               // 추상메소드 선언
                public void display(String s);
}

class StarDisplay2 implements DisplayInterface
{               // 추상메소드 구현
                public void display(String s)
                {               // 문자열 좌우에 별표 출력
                                System.out.print("*** ");
                                System.out.print(s);
                                System.out.println(" ***");
                }
};
class BarDisplay2 implements DisplayInterface
{               // 추상메소드 구현
                public void display(String s)
                {               // 문자열 좌우에 bar 표시 출력
                                System.out.print("||| ");
                                System.out.print(s);
                                System.out.println(" |||");
                }
};
```

```
public class InterfaceDisplay
{
    public static void main(String arg[])
    {
                // 객체 변수를 이용한 출력
                StarDisplay2 si = new StarDisplay2();
                si.display("글자 인쇄 시험 1");

                BarDisplay2 bi = new BarDisplay2();
                bi.display("글자 인쇄 시험 2");

                // 인터페이스 변수를 이용한 출력
                DisplayInterface di1 = new StarDisplay2();
                di1.display("글자 인쇄 시험 3");

                DisplayInterface di2 = new BarDisplay2();
                di2.display("글자 인쇄 시험 4");
    }
};
```

인터페이스와 다형성

인터페이스 타입

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

```
인터페이스타입 변수 = 구현객체
```

구현 객체는 인터페이스 타입이므로 자동 변환된다.

- 인터페이스 타입 변수가 구현 객체를 참조한다면 강제 타입 변환 가능

```
구현클래스타입 변수 = (구현클래스타입) 인터페이스타입변수
```

타입 변환 연산자이다.

인터페이스 구현 객체를 참조하는 변수이다.

인터페이스와 다형성

- 타입 변환과 다형성

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

```
<terminated> ControllableDemo [
Turn on TV
Turn off TV
Repair a device
Turn on computer
Turn off computer
Repair a device
Reset a device
```

인터페이스와 다형성

- 타입 변환과 다형성

```
1
2  public class ControllableDemo {
3      public static void main(String[] args) {
4          Controllable[] controllable = { new TV(), new Computer() };
5
6          for (Controllable c : controllable) {
7              c.turnOn();
8              c.turnOff();
9              c.repair();
10         }
11         Controllable.reset();
12     }
13 }
```

```
<terminated> ControllableDe
Turn on TV
Turn off TV
Repair a device
Turn on computer
Turn off computer
Repair a device
Reset a device
```

인터페이스와 다형성

타입 변환과 다형성

```
1  interface Animal {
2      void sound();
3  }
4
5  class Dog implements Animal {
6      public void sound() {
7          System.out.println("Dog sound");
8      }
9  }
10
11 class Cuckoo implements Animal {
12     public void sound() {
13         System.out.println("Cuckoo sound ");
14     }
15 }
16
17 public class AnimalDemo {
18     public static void main(String[] args) {
19         Dog d = new Dog();
20         Cuckoo c = new Cuckoo();
21
22         makeSound(d);
23         makeSound(c);
24     }
25
26     static void makeSound(Animal a) {
27         a.sound();
28     }
29 }
```

```
<terminated> AnimalD
Dog sound
Cuckoo sound
```

Q & A