



8장. 인터페이스

이것이 자바다 (<http://cafe.naver.com/thisjava>)

Contents

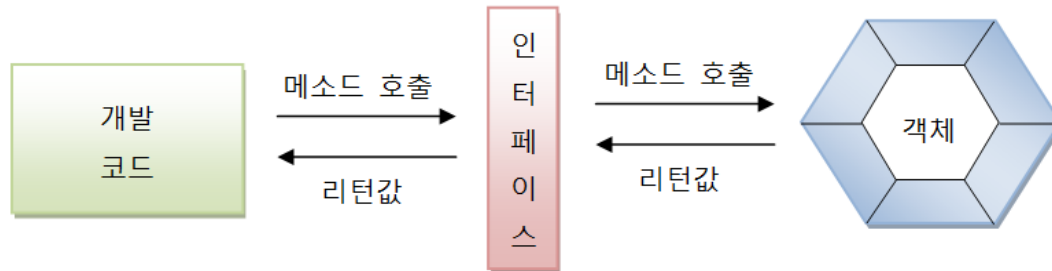
- ❖ 1절. 인터페이스의 역할
- ❖ 2절. 인터페이스 선언
- ❖ 3절. 인터페이스 구현
- ❖ 4절. 인터페이스 사용
- ❖ 5절. 타입변환과 다형성
- ❖ 6절. 인터페이스 상속
- ❖ 7절. 디폴트 메소드와 인터페이스 확장



1절. 인터페이스의 역할

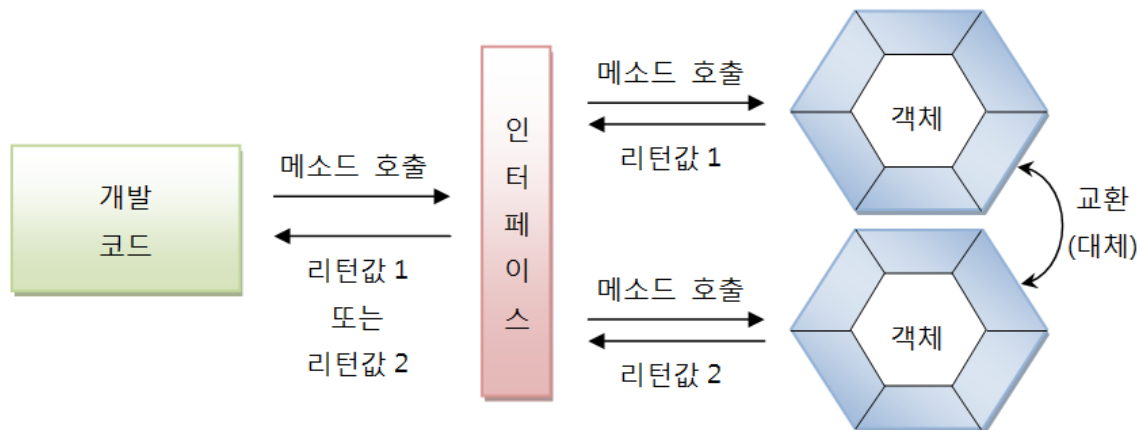
❖ 인터페이스란?

- 개발 코드와 객체가 서로 통신하는 접점
 - 개발 코드는 **인터페이스의 메소드만 알고 있으면 OK**



■ 인터페이스의 역할

- 개발 코드가 객체에 종속되지 않게 -> 객체 교체할 수 있도록 하는 역할
- 개발 코드 변경 없이 리턴값 또는 실행 내용이 다양해 질 수 있음 (다형성)



2절. 인터페이스 선언

❖ 인터페이스 선언

- 인터페이스 이름 - 자바 식별자 작성 규칙에 따라 작성
- 소스 파일 생성
 - 인터페이스 이름과 대소문자가 동일한 소스 파일 생성
- 인터페이스 선언

```
[ public ] interface 인터페이스명 { ... }
```



2절. 인터페이스 선언

❖ 인터페이스 선언

■ 인터페이스의 구성 멤버 (p.346)

```
interface 인터페이스명 {  
    //상수  
    타입 상수명 = 값;  
    //추상 메소드  
    타입 메소드명(매개변수,...);  
    //디폴트 메소드  
    default 타입 메소드명(매개변수,...) {...}  
    //정적 메소드  
    static 타입 메소드명(매개변수) {...}  
}
```



2절. 인터페이스 선언

❖ 상수 필드 선언

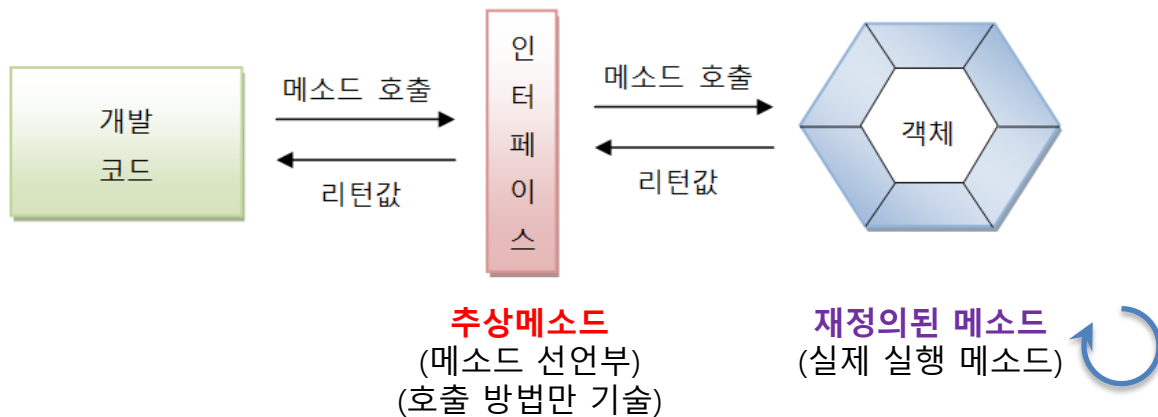
- 인터페이스는 상수 필드만 선언 가능
 - 데이터 저장하지 않음
- 인터페이스에 선언된 필드는 모두 `public static final`
 - 자동적으로 컴파일 과정에서 붙음
- 상수명은 대문자로 작성
 - 서로 다른 단어로 구성되어 있을 경우에는 언더 바(_)로 연결
- 선언과 동시에 초기값 지정
 - `static { }` 블록 작성 불가 - `static { }` 으로 초기화 불가



2절. 인터페이스 선언

❖ 추상 메소드 선언

- 인터페이스 통해 호출된 메소드는 최종적으로 객체에서 실행
 - 인터페이스의 메소드는 기본적으로 실행 블록이 없는 추상 메소드로 선언
 - `public abstract`를 생략하더라도 자동적으로 컴파일 과정에서 붙게 됨



[`public abstract`] 리턴타입 메소드명(매개변수, ...);



2절. 인터페이스 선언

❖ 디폴트 메소드 선언

- 자바8에서 추가된 인터페이스의 새로운 멤버

```
[public] default 리턴타입 메소드명(매개변수, ...) { ... }
```

- 실행 블록을 가지고 있는 메소드
- default 키워드를 반드시 붙여야
- 기본적으로 public 접근 제한
 - 생략하더라도 컴파일 과정에서 자동 붙음



2절. 인터페이스 선언

❖ 정적 메소드 선언

- 자바8에서 추가된 인터페이스의 새로운 멤버

```
[public] static 리턴타입 메소드명(매개변수, ...) { ... }
```

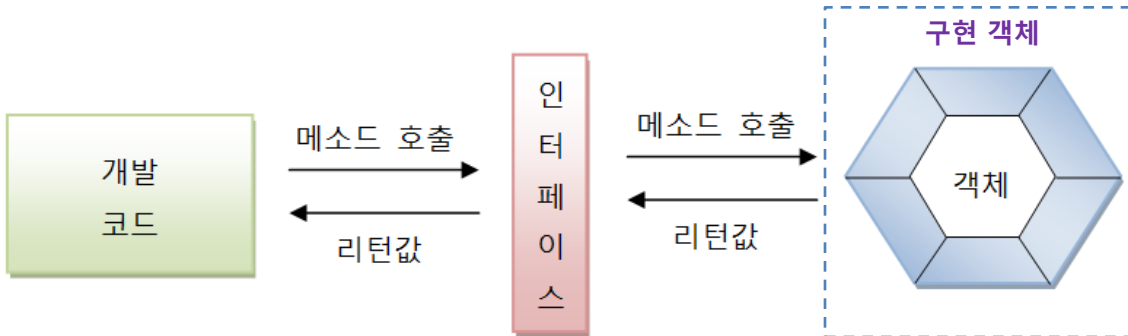
```
public interface RemoteControl {  
    static void changeBattery() {  
        System.out.println("건전지를 교환합니다.");  
    }  
}
```



3절. 인터페이스 구현

❖ 구현 객체와 구현 클래스

- 인터페이스의 추상 메소드 대한 실제 메소드를 가진 객체 = 구현 객체



- 구현 객체를 생성하는 클래스 = 구현 클래스



3절. 인터페이스 구현

❖ 구현 클래스 선언

- 자신의 객체가 인터페이스 타입으로 사용할 수 있음
 - implements 키워드로 명시

```
public class 구현클래스명 implements 인터페이스명 {  
    //인터페이스에 선언된 추상 메소드의 실제 메소드 선언  
}
```

❖ 추상 메소드의 실제 메소드를 작성하는 방법

- 메소드의 선언부가 정확히 일치해야
- 인터페이스의 모든 추상 메소드를 재정의하는 실제 메소드 작성해야
 - 일부만 재정의할 경우, 추상 클래스로 선언 + abstract 키워드 붙임



3절. 인터페이스 구현

❖ 익명 구현 객체

■ 명시적인 구현 클래스 작성 생략하고 바로 구현 객체를 얻는 방법

- 이름 없는 구현 클래스 선언과 동시에 객체 생성

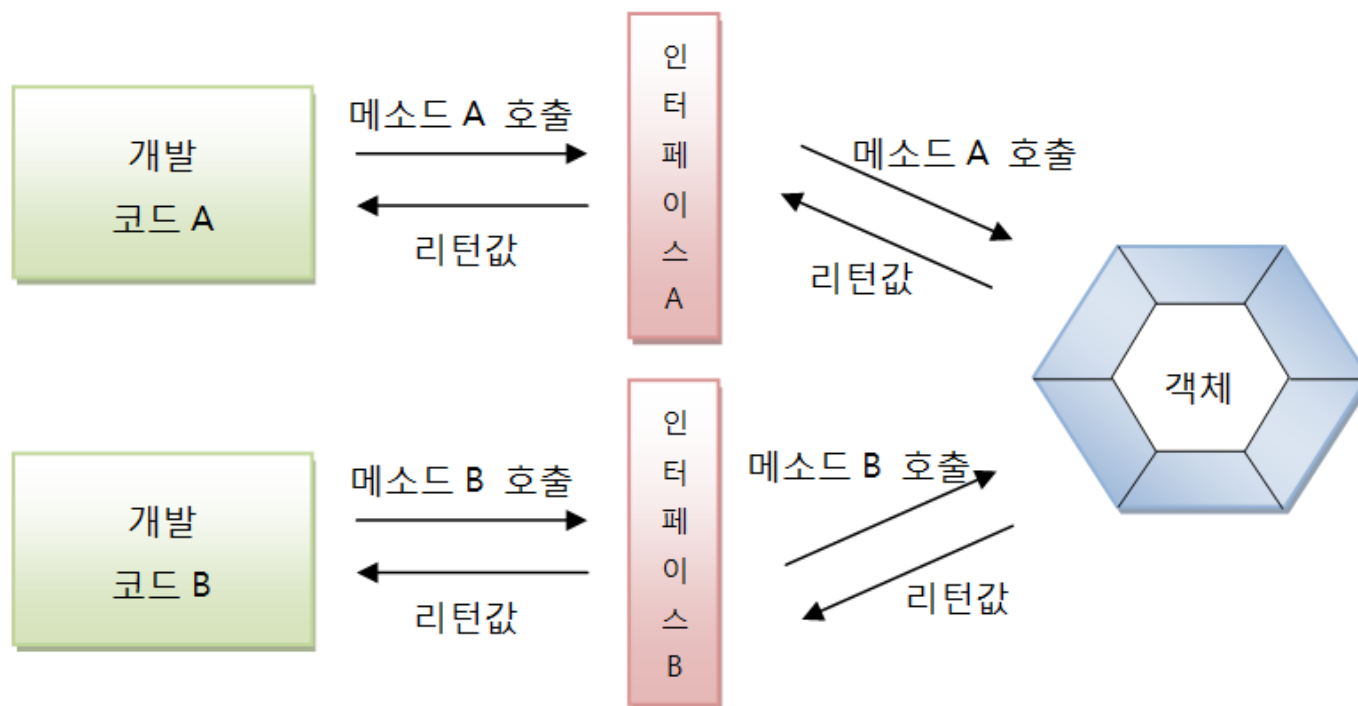
```
인터페이스 변수 = new 인터페이스() {  
    //인터페이스에 선언된 추상 메소드의 실제 메소드 선언  
};
```

- 인터페이스의 추상 메소드들을 모두 재정의하는 실제 메소드가 있어야
- 추가적으로 필드와 메소드 선언 가능하나 익명 객체 안에서만 사용
- 인터페이스 변수로 접근 불가



3절. 인터페이스 구현

❖ 다중 인터페이스 구현 클래스



```
public class 구현클래스명 implements 인터페이스 A, 인터페이스 B {  
    //인터페이스 A 에 선언된 추상 메소드의 실제 메소드 선언  
    //인터페이스 B 에 선언된 추상 메소드의 실제 메소드 선언  
}
```



4절. 인터페이스 사용

❖ 인터페이스에 구현 객체를 대입하는 방법 (p.357~358)

```
인터페이스 변수;  
변수 = 구현객체;
```

```
인터페이스 변수 = 구현객체;
```

```
RemoteControl rc;  
rc = new Television();  
rc = new Audio();
```



4절. 인터페이스 사용

❖ 추상 메소드 사용

```
RemoteControl rc = new Television();
```

```
rc.turnOn();    → Television 의 turnOn() 실행
```

```
rc.turnOff();   → Television 의 turnOff() 실행
```



4절. 인터페이스 사용

❖ 디폴트 메소드 사용

- 인터페이스만으로는 사용 불가

- 구현 객체가 인터페이스에 대입되어야 호출할 수 있는 인스턴스 메소드

- 모든 구현 객체가 가지고 있는 기본 메소드로 사용

- 필요에 따라 구현 클래스가 디폴트 메소드 재정의해 사용

❖ 정적 메소드 사용

- 인터페이스로 바로 호출 가능



5절. 타입변환과 다형성

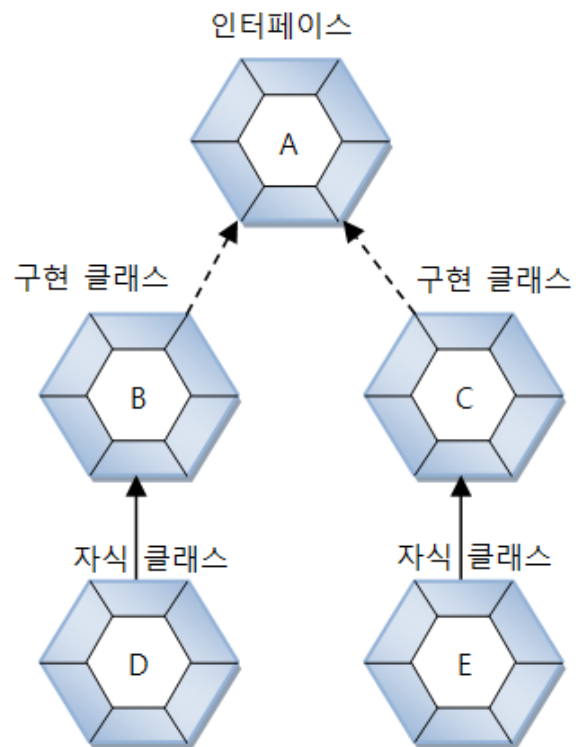
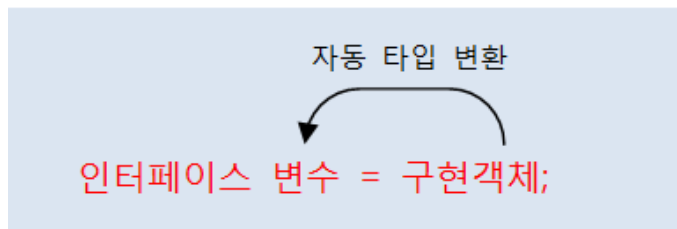
❖ 다형성 (p.362~364)

- 하나의 타입에 여러 가지 객체 대입해 다양한 실행 결과를 얻는 것
- 다형성을 구현하는 기술
 - 상속 또는 인터페이스의 자동 타입 변환(Promotion)
 - 오버라이딩(Overriding)
- 다형성의 효과
 - 다양한 실행 결과를 얻을 수 있음
 - 객체를 부품화시킬 수 있어 유지보수 용이 (메소드의 매개변수로 사용)



5절. 타입변환과 다형성

❖ 자동 타입 변환(Promotion)



```
B b = new B();  
C c = new C();  
D d = new D();  
E e = new E();
```



```
A a1 = b; (가능)  
A a2 = c; (가능)  
A a3 = d; (가능)  
A a4 = e; (가능)
```



5절. 타입변환과 다형성

❖ 필드의 다형성 (p.365~368)

[다형성은 객체를 부품화시킨다]



```
public interface Tire {  
    public void roll();  
}
```

```
public class HankookTire implements Tire {  
    @Override  
    public void roll() {  
        System.out.println("한국 타이어가 굴러갑니다.");  
    }  
}
```

```
public class Car {  
    Tire frontLeftTire = new HankookTire();  
    Tire frontRightTire = new HankookTire();  
    Tire backLeftTire = new HankookTire();  
    Tire backRightTire = new HankookTire();  
  
    void run() {  
        frontLeftTire.roll();  
        frontRightTire.roll();  
        backLeftTire.roll();  
        backRightTire.roll();  
    }  
}
```

```
Car myCar = new Car();  
myCar.frontLeftTire = new KumhoTire();  
myCar.frontRightTire = new KumhoTire();
```

```
myCar.run();
```

5절. 타입변환과 다형성

❖ 인터페이스 배열로 구현한 객체 관리

```
Tire[] tires = {  
    new HankookTire(),  
    new HankookTire(),  
    new HankookTire(),  
    new HankookTire()  
};
```

```
tires[1] = new KumhoTire();
```

```
void run() {  
    for(Tire tire : tires) {  
        tire.roll();  
    }  
}
```



5절. 타입변환과 다형성

❖ 매개변수의 다형성

- 매개 변수의 타입이 인터페이스인 경우
 - 어떠한 구현 객체도 매개값으로 사용 가능
 - 구현 객체에 따라 메소드 실행결과 달라짐

❖ 강제 타입 변환(Casting)

- 인터페이스 타입으로 자동 타입 변환 후, 구현 클래스 타입으로 변환
 - 필요성: 구현 클래스 타입에 선언된 다른 멤버 사용하기 위해

❖ 객체 타입 확인(instanceof 연산자) (p.375~377)

- 강제 타입 변환 전 구현 클래스 타입 조사



6절. 인터페이스 상속

❖ 인터페이스간 상속 가능

```
public interface 하위인터페이스 extends 상위인터페이스 1, 상위인터페이스 2 { ... }
```

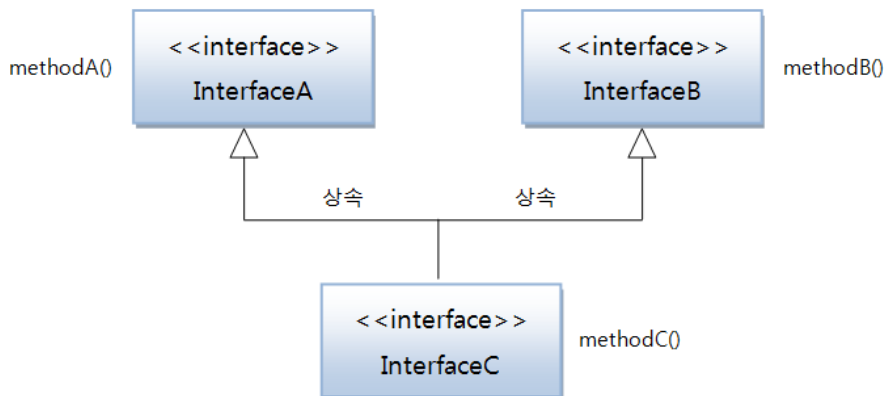
■ 하위 인터페이스 구현 클래스는 아래 추상 메소드를 모두 재정의해야

- 하위 인터페이스의 추상 메소드
- 상위 인터페이스1의 추상 메소드
- 상위 인터페이스2의 추상 메소드

```
하위인터페이스 변수 = new 구현클래스(...);  
상위인터페이스 1 변수 = new 구현클래스(...);  
상위인터페이스 2 변수 = new 구현클래스(...);
```

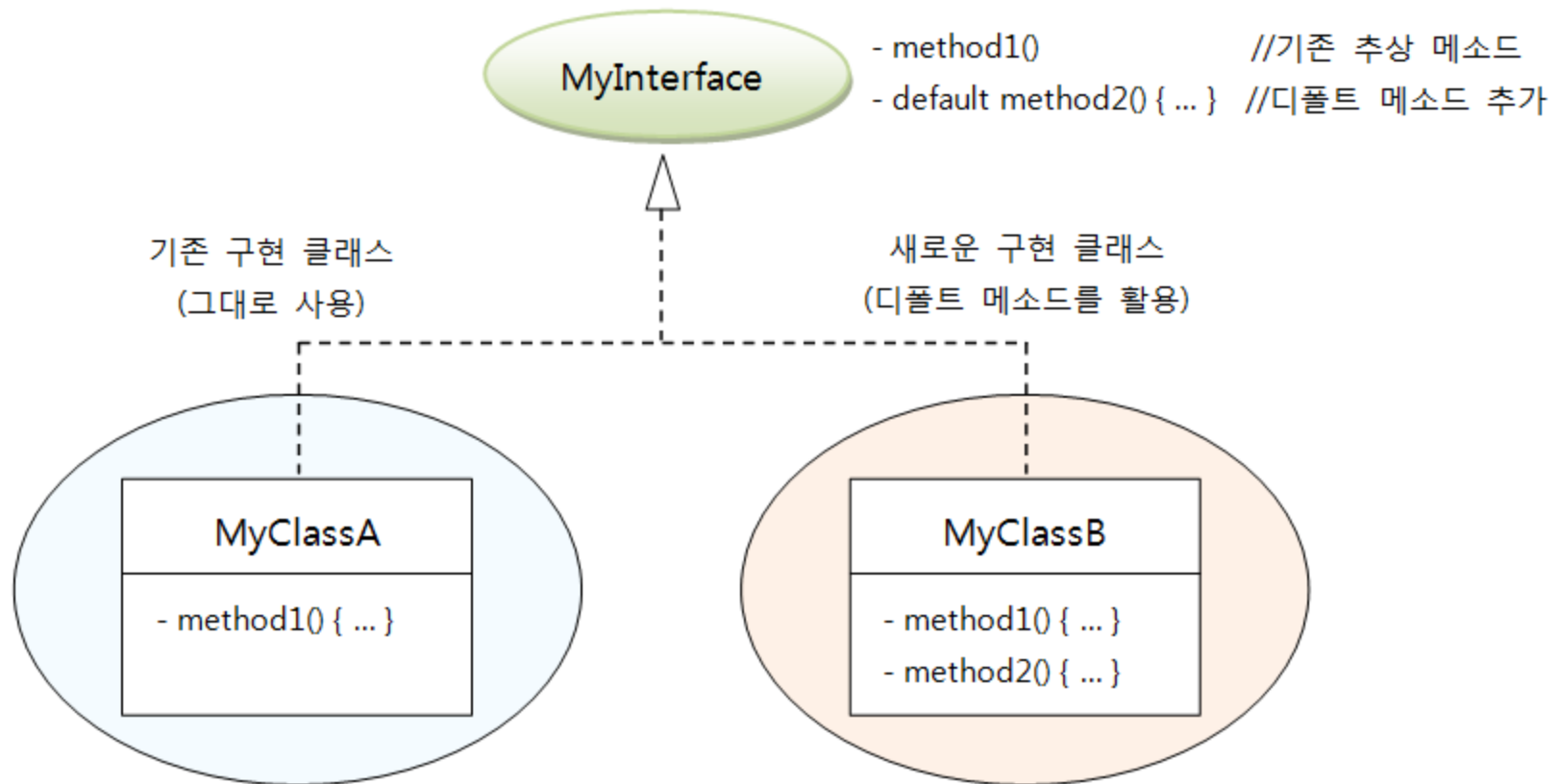
■ 인터페이스 자동 타입 변환

- 해당 타입의 인터페이스에 선언된 메소드만 호출 가능



7절. 디폴트 메소드와 인터페이스 확장

❖ 디폴트 메소드와 확장 메소드 사용하기 (p.379~382)



7절. 디폴트 메소드와 인터페이스 확장

❖ 디폴트 메소드가 있는 인터페이스 상속

- 부모 인터페이스의 디폴트 메소드를 자식 인터페이스에서 활용 방법
 - 디폴트 메소드를 단순히 상속만 받음
 - 디폴트 메소드를 재정의(Override)해서 실행 내용을 변경
 - 디폴트 메소드를 추상 메소드로 재선언

