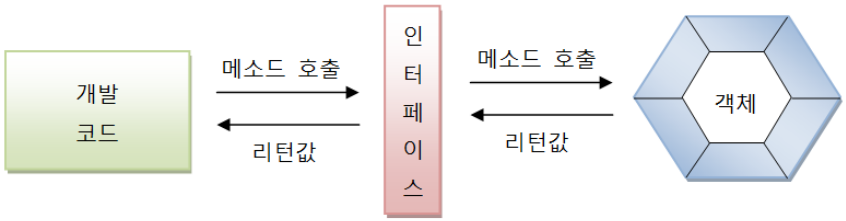
인터페이스

Contents

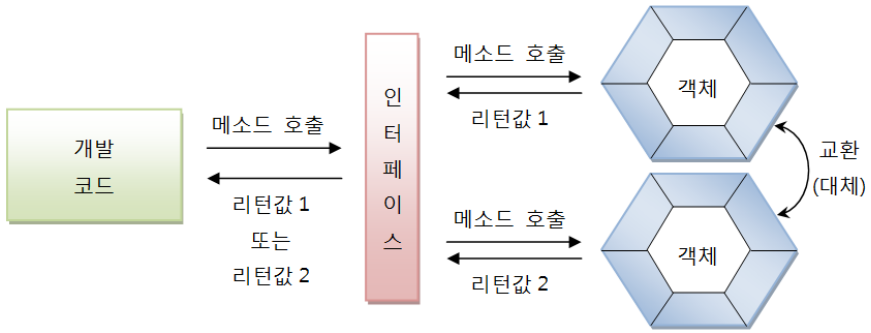
1절. 인터페이스의 역할

❖ 인터페이스란?

- 객체의 사용 방법을 정의한 타입
- 객체의 교환성을 높여주기 때문에 다형성을 구현하는 매우 중요한 역할
- 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

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

- ❖ 상수 필드
- ❖ 추상 메소드
- ❖ 디폴트 메소드
- ❖ 정적 메소드

※ 인터페이스는 객체 사용 설명서이므로 런타임 시 데이터를 저장할 수 있는 필드를 선언할 수 없다

2절. 인터페이스 선언

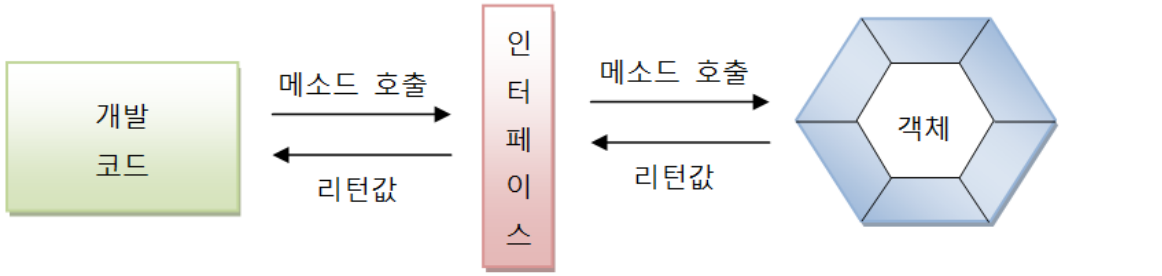
❖ 상수 필드 선언

- 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

2절. 인터페이스 선언

❖ 추상 메소드 선언

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

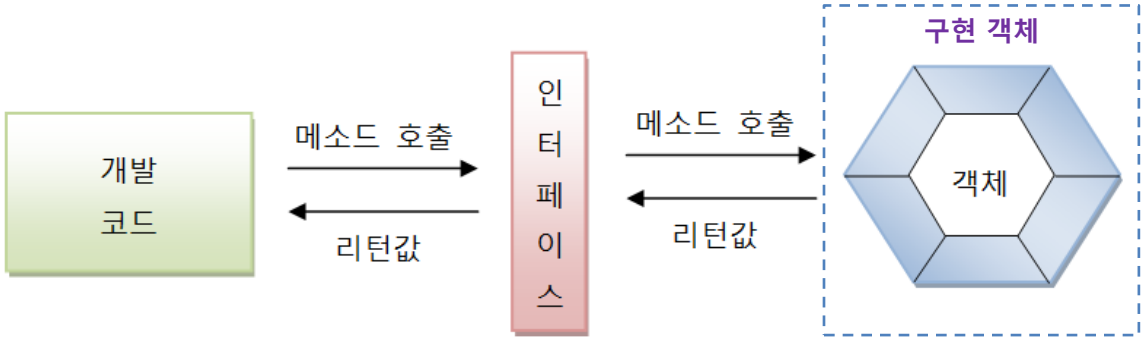


[public abstract] 리턴타입 메소드명(매개변수, ...);

3절. 인터페이스 구현

❖ 구현 객체와 구현 클래스

- 인터페이스의 추상 메소드에 대한 실체 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 실체 메소드 선언
}
```

❖ 추상 메소드의 실체 메소드를 작성하는 방법

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

3절. 인터페이스 구현

❖ 인터페이스 선언 예제 (New > interface > RemoteControl.java)

```
public interface RemoteControl {

    //인터페이스에 선언된 메소드는 실행문이 없는 추상 메소드
    //public abstract를 생략해도 컴파일 과정에서 자동으로 생성
    //최종적으로 구현 클래스에서 정의되고 객체에서 실행
    public void turnOn();
}
```

3절. 인터페이스 구현

❖ 구현 클래스 선언 예제

```
public class TV implements RemoteControl{

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

}
```

RemoteControl 인터페이스를 구현하는 구현 클래스

```
public class Audio implements RemoteControl{

    @Override
    public void turnOn() {
        System.out.println("오디오를 켭니다.");
    }

}
```

3절. 인터페이스 구현

❖ 실행 클래스 (RemoteControlEx.java)

```
public class RemoteControlEx {
    public static void main(String[] args) {
        //구현 클래스를 이용한 객체 생성
        Audio audio  = new Audio() ;
        TV    tv     = new TV();

        //생성된 객체를 이용한 메소드 호출
        audio.turnOn();
        tv.turnOn();
    }
}
```

3절. 인터페이스 구현

❖ 인터페이스-상수 필드

```
public interface RemoteControl {
    //인터페이스는 객체 사용 설명서이므로 실행 도중 데이터를 저장하는 필드를 선언할 수 없다.
    //인터페이스에서 선언 가능한 것은 상수(static final) 이다.
    //상수를 선언할 때는 반드시 초기값을 설정해야 한다.
    //static final을 생략해도 컴파일 과정에서 자동 생성된다.
    int MAX_VOLUME = 10;
    int MIN_VOLUME = 0;

    //인터페이스에 선언된 메소드는 실행문이 없는 추상 메소드
    //public abstract를 생략해도 컴파일 과정에서 자동으로 생성
    //최종적으로 구현 클래스에서 정의되고 객체에서 실행
    public void turnOn();
}
```

3절. 인터페이스 구현

❖ 인터페이스-상수 필드

```
public class RemoteControlEx {
    public static void main(String[] args) {
        //구현 클래스를 이용한 객체 생성
        Audio audio = new Audio() ;
        TV    tv    = new TV();

        //생성된 객체를 이용한 메소드 호출
        audio.turnOn();
        tv.turnOn();

        System.out.println("리모콘 최대 볼륨 : " + RemoteControl.MAX_VOLUME);
        System.out.println("리모콘 최소 볼륨 : " + RemoteControl.MIN_VOLUME);
    }
}
```

3절. 인터페이스 구현

❖ 인터페이스-추상 메소드 추가

```
public interface RemoteControl {
    //인터페이스는 객체 사용 설명서이므로 실행 도중 데이터를 저장하는 필드를 선언할 수 없다.
    //인터페이스에서 선언 가능한 것은 상수(static final) 이다.
    //상수를 선언할 때는 반드시 초기값을 설정해야 한다.
    //static final을 생략해도 컴파일 과정에서 자동 생성된다.
    int MAX_VOLUME = 10;
    int MIN_VOLUME = 0;

    //인터페이스에 선언된 메소드는 실행문이 없는 추상 메소드
    //public abstract를 생략해도 컴파일 과정에서 자동으로 생성
    //최종적으로 구현 클래스에서 정의되고 객체에서 실행
    public void turnOn();
    void turnOff();
    void setVolume(int volume);
}
```

3절. 인터페이스 구현

❖ 인터페이스-추상 메소드 재정의

```java
public class TV implements RemoteControl{
    private int volume;

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

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

    @Override
    public void setVolume(int volume) {
        this.volume = volume;
    }

}
```

```java
public class Audio implements RemoteControl{
    private int volume;

    @Override
    public void turnOn() {
        System.out.println("오디오를 켭니다.");
    }

    @Override
    public void turnOff() {
        System.out.println("오디오를 끕니다.");
    }

    @Override
    public void setVolume(int volume) {
        this.volume = volume;
    }

}
```

3절. 인터페이스 구현

❖ 인터페이스-추상 메소드 재정의

- 인터페이스 상수 필드를 이용해서 volume 필드의 유효성 검사

```
@Override
public void setVolume(int volume) {
    if (volume > RemoteControl.MAX_VOLUME)
        this.volume = RemoteControl.MAX_VOLUME;
    else if (volume < RemoteControl.MIN_VOLUME)
        this.volume = RemoteControl.MIN_VOLUME;
    else
        this.volume = volume;
}
```

```
System.out.println("현재 TV 볼륨 : " + volume);
```

3절. 인터페이스 구현

❖ 인터페이스-실행 클래스

```
public class RemoteControlEx {
    public static void main(String[] args) {
        //구현 클래스를 이용한 객체 생성
        Audio audio = new Audio();
        TV    tv    = new TV();

        //생성된 객체를 이용한 메소드 호출
        audio.turnOn();
        audio.setVolume(5);

        audio.turnOff();

        System.out.println("------");

        tv.turnOn();
        tv.setVolume(3);

        tv.turnOff();

        System.out.println("리모콘 최대 볼륨 : " + RemoteControl.MAX_VOLUME);
        System.out.println("리모콘 최소 볼륨 : " + RemoteControl.MIN_VOLUME);
    }
}
```

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 한다
- 기본적으로 public 접근 제한의 특성을 가짐
  - 생략하더라도 컴파일 과정에서 자동 붙음
- 인터페이스의 모든 구현 객체가 가지고 있는 기본 메소드로, 반드시 구현 객체가 있어야 실행 가능한 메소드이다

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.");
    }
}
```

- 객체가 없어도 인터페이스만으로 호출이 가능하다

```
RemoteControl.changeBattery();
```

2절. 인터페이스 선언

❖ 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

3절. 인터페이스 구현

❖ 인터페이스-디폴트 메소드

```
public interface RemoteControl {
    int MAX_VOLUME = 10;
    int MIN_VOLUME = 0;

    public void turnOn();
    void turnOff();
    void setVolume(int volume);

    //디폴트 메소드 선언 => 기존 구현 클래스에는 영향을 주지 않는다.
    default void setMute(boolean mute) {
        if (mute) {
            System.out.println("무음 처리 합니다.");
            setVolume(MIN_VOLUME); //추상 메소드 호출(상수 사용)
        } else {
            System.out.println("무음 해제 합니다.");
        }
    }
}
```

3절. 인터페이스 구현

❖ 인터페이스-디폴트 메소드

```
public class RemoteControlEx {
    public static void main(String[] args) {
        //구현 클래스를 이용한 객체 생성
        Audio audio = new Audio();
        TV    tv    = new TV();

        //생성된 객체를 이용한 메소드 호출
        audio.turnOn();
        audio.setVolume(5);
        audio.setMute(true);
        audio.turnOff();

        System.out.println("------");

        tv.turnOn();
        tv.setVolume(3);
        tv.setMute(true);
        tv.turnOff();

        System.out.println("리모콘 최대 볼륨 : " + RemoteControl.MAX_VOLUME);
        System.out.println("리모콘 최소 볼륨 : " + RemoteControl.MIN_VOLUME);
    }
}
```

3절. 인터페이스 구현

❖ 구현 클래스-디폴트 메소드 재정의(Audio.java)

```
private int memoryVol;

@Override
public void setMute(boolean mute) {
    if (mute) {
        this.memoryVol = this.volume;
        System.out.println("Audio 무음 처리...");
        setVolume(RemoteControl.MIN_VOLUME);
    } else {
        System.out.println("Audio 무음 해제~");

        //무음 해제일 때 원래 볼륨으로 복원
        setVolume(memoryVol);
    }
}
```

3절. 인터페이스 구현

❖ 구현 클래스-디폴트 메소드 재정의(Audio.java)

```
private int memoryVol;

@Override
public void setMute(boolean mute) {
    if (mute) {
        this.memoryVol = this.volume;
        System.out.println("Audio 무음 처리
        setVolume(RemoteControl.MIN_VOLU
    } else {
        System.out.println("Audio 무음 해제

        //무음 해제일 때 원래 볼륨으로 복원
        setVolume(memoryVol);
    }
}
```

```
오디오를 켭니다.
현재 Audio 볼륨 : 5
Audio 무음 처리...
현재 Audio 볼륨 : 0
Audio 무음 해제~
현재 Audio 볼륨 : 5
오디오를 끕니다.
------
TV를 켭니다.
현재 TV 볼륨 : 3
무음 처리 합니다.
현재 TV 볼륨 : 0
TV를 끕니다.
```

3절. 인터페이스 구현

❖ 익명 구현 객체

- 명시적인 구현 클래스 작성은 생략하고 바로 구현 객체를 얻는 방법
  - 이름 없는 구현 클래스 선언과 동시에 객체 생성

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

  - 인터페이스의 추상 메소드들을 모두 재정의하는 실체 메소드가 있어야 함
  - 추가적으로 필드와 메소드 선언 가능하나 익명 객체 안에서만 사용
  - 인터페이스 변수로 접근 불가
- 활용
  - UI 프로그래밍에서 이벤트 처리 시
  - 임시 작업 스레드를 만들기 위해 활용
  - 자바 8에서 지원하는 람다식은 인터페이스의 익명 구현 객체를 사용

3절. 인터페이스 구현

❖ 익명 구현 객체 실행 클래스 (VolumeControlEx.java)

Volume.java | Radio.java | TV.java | Speaker.java | VolumeEx.java | VolumeCont

```
1  package week12;
2
3  public class VolumeControlEx {
4      public static void main(String[] args) {
5          Volume vol = new Volume() {
6              @Override
7              public void volumeUp(int level) {
8                  System.out.println("익명 객체 볼륨을 올립니다 : "+level);
9              }
10
11             @Override
12             public void volumeDown(int level) {
13                 System.out.println("익명 객체 볼륨을 내립니다 : "+level);
14             }
15         };
16
17         vol.volumeUp(5);
18         vol.volumeDown(3);
19     }
20 }
```

3절. 인터페이스 구현

❖ 익명 구현 객체 실행 클래스 (VolumeControlEx.java)

Volume.java | Radio.java | TV.java | Speaker.java | VolumeEx.java | VolumeContr

```
1  package week12;
2
3  public class VolumeControlEx {
4      public static void main(String[] args) {
5          Volume vol = new Volume() {
6              @Override
7              public void volumeUp(int level) {
8                  System.out.println("익명 객체 볼륨을 올립니다 : "+level);
9              }
10
11             @Override
12             public void volumeDown(int level) {
13                 System.out.println("익명 객체 볼륨을 내립니다 : "+level);
14             }
15         };
16
17         vol.volumeUp(5);
18         vol.volumeDown(3);
19     }
20 }
```

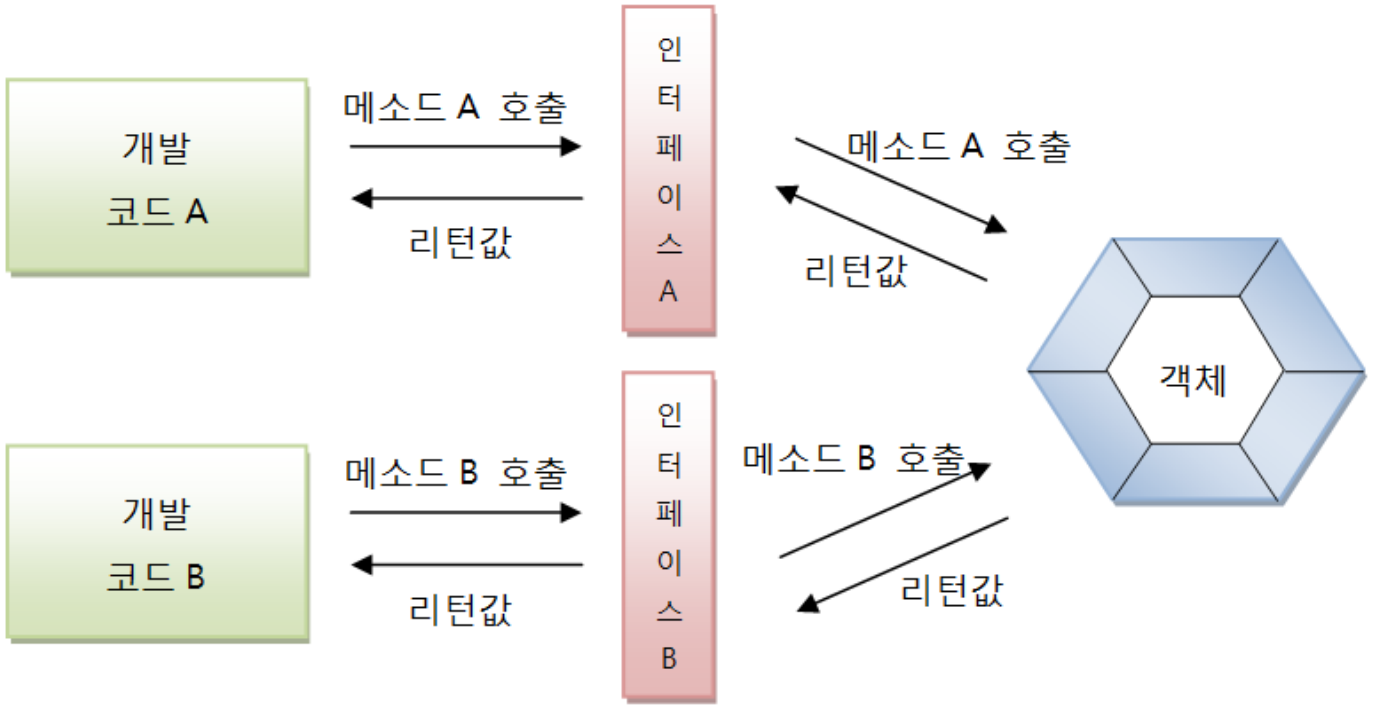
Console

<terminated> VolumeControlEx [Java Ap

```
익명 객체 볼륨을 올립니다 : 5
익명 객체 볼륨을 내립니다 : 3
```

3절. 인터페이스 구현

❖ 다중 인터페이스 구현 클래스




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

3절. 인터페이스 구현

❖ 다중 인터페이스 구현 클래스

Volume.java | Searchable.java | SmartTV.java | SmartTVEx.java

```java
package week12;

public interface Searchable {
    void search(String url);
}
```

Searchable.java
(인터페이스)

```java
public class SmartTV implements RemoteControl, Searchable{
    private int volume;

    @Override
    public void search(String url) {
        System.out.println(url + "을 검색합니다.");
    }

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

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

SmartTV.java
(클래스)

3절. 인터페이스 구현

❖ 다중

```
    @Override
    public void setVolume(int volume) {
        if (volume > RemoteControl.MAX_VOLUME)
            this.volume = RemoteControl.MAX_VOLUME;
        else if (volume < RemoteControl.MIN_VOLUME)
            this.volume = RemoteControl.MIN_VOLUME;
        else
            this.volume = volume;

        System.out.println("현재 Smart TV 볼륨 : " + volume);
    }

    private int memoryVol;

    @Override
    public void setMute(boolean mute) {
        if (mute) {
            this.memoryVol = this.volume;
            System.out.println("Smart TV 무음 처리...");
            setVolume(RemoteControl.MIN_VOLUME);
        } else {
            System.out.println("Smart TV 무음 해제~");

            //무음 해제일 때 원래 볼륨으로 복원
            setVolume(memoryVol);
        }
    }
}
```

SmartTV.java
(클래스)

3절. 인터페이스 구현

❖ 다중 인터페이스 구현 클래스 (RemoteControlEx.java)

```
SmartTV smart = new SmartTV();
smart.turnOn();
smart.setVolume(3);
smart.search("네이버");
smart.setMute(true);
smart.setMute(false);
smart.turnOff();
```

```
Smart TV를 켭니다.
현재 Smart TV 볼륨 : 3
네이버을 검색합니다.
Smart TV 무음 처리...
현재 Smart TV 볼륨 : 0
Smart TV 무음 해제~
현재 Smart TV 볼륨 : 3
Smart TV를 끕니다.
```

4절. 인터페이스 사용

❖ 인터페이스에 구현 객체를 대입하는 방법

```
인터페이스 변수;
변수 = 구현객체;
```

```
인터페이스 변수 = 구현객체;
```

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

```
RemoteControl rc = new Audio();

RemoteControl rc = new TV();
```

4절. 인터페이스 사용

❖ 인터페이스 사용

```
RemoteControl rc = new Television();
rc.turnOn();    → Television 의 turnOn() 실행
rc.turnOff();   → Television 의 turnOff() 실행
```

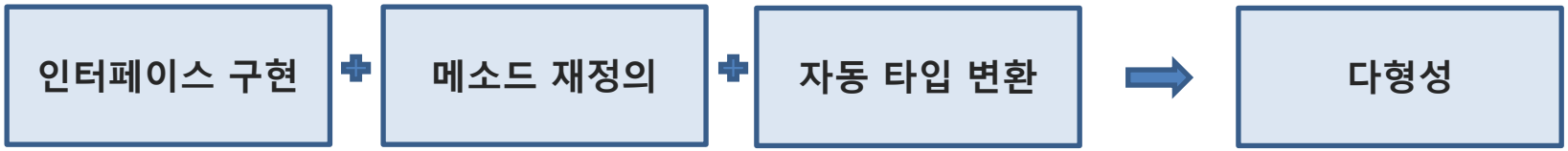
5절. 타입변환과 다형성

❖ 다형성

- 하나의 타입에 여러 가지 객체를 대입해 다양한 실행 결과를 얻는 것

- 다형성을 구현하는 기술
  - 상속 또는 인터페이스의 자동 타입 변환(Promotion)
  - 오버라이딩(Overriding)

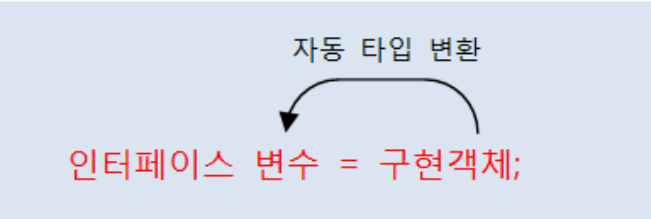
- 다형성의 효과
  - 다양한 실행 결과를 얻을 수 있음
  - 객체를 부품화시킬 수 있어 유지보수 용이 (메소드의 매개변수로 사용)

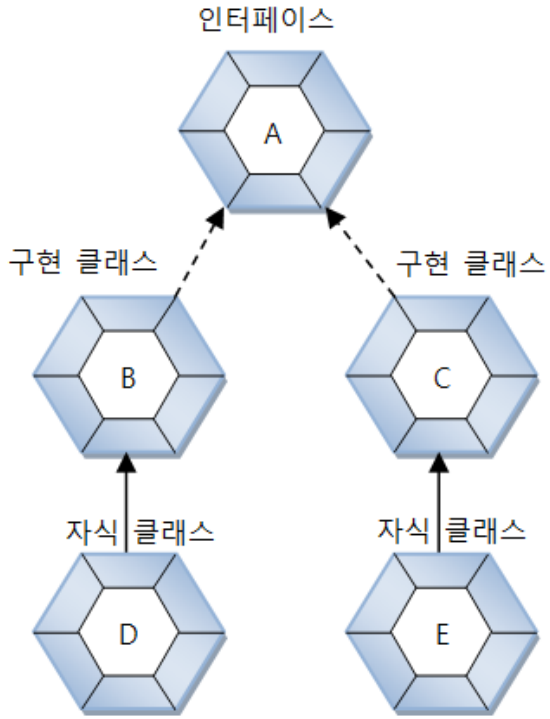


5절. 타입변환과 다형성

❖ 인터페이스 자동 타입 변환(Promotion)

- 인터페이스와 구현 클래스 간에 발생



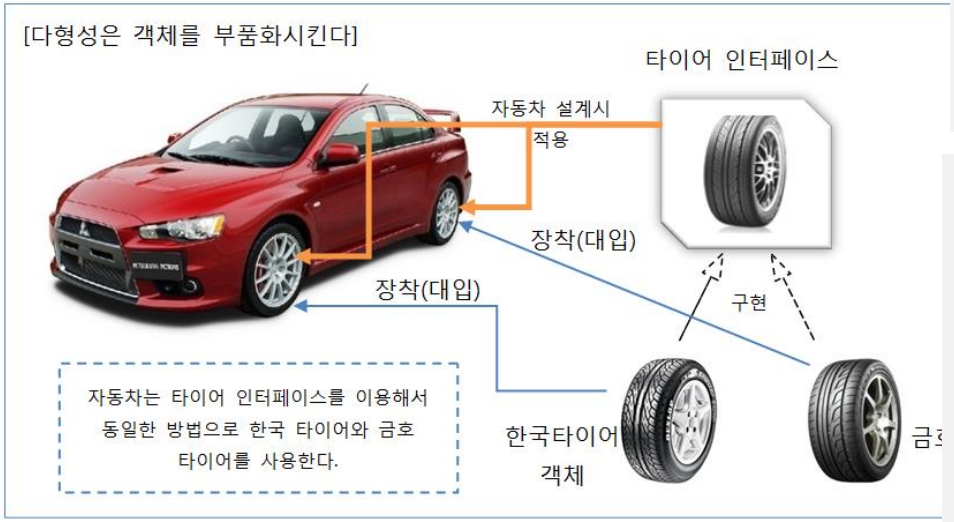



```
B b = new B();
C c = new C();
D d = new D();
E e = new E();
```

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

5절. 타입변환과 다형성

❖ 필드의 다형성




```
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

❖ 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)

- 강제 타입 변환 전 구현 클래스 타입 조사

6절. 인터페이스 상속

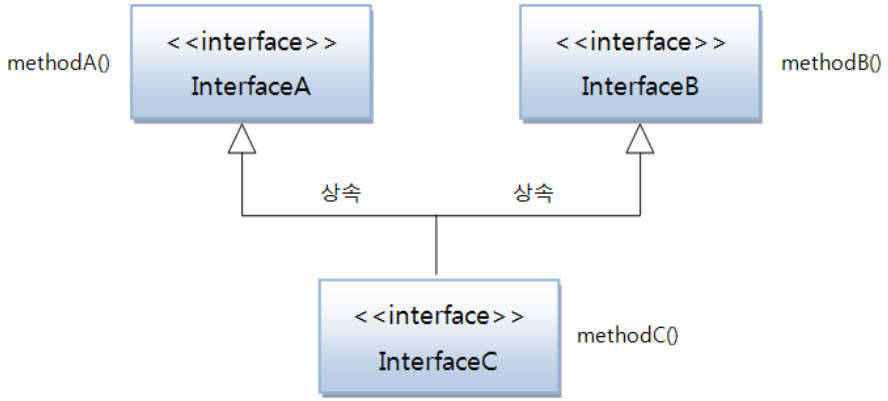
❖ 인터페이스간 상속 가능

```
public interface 하위인터페이스 extends 상위인터페이스1, 상위인터페이스2 { ... }
```

- 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(...);
상위인터페이스1 변수 = new 구현클래스(...);
상위인터페이스2 변수 = new 구현클래스(...);
```

- 인터페이스 자동 타입 변환
  - 해당 타입의 인터페이스에 선언된 메소드만 호출 가능



3절. 인터페이스 구현

❖ 실행 클래스 (RemoteControlEx.java)

```
System.out.println("------\n << 다형성 결과 >>\n");

RemoteControl[] rc = new RemoteControl[3];
rc[0] = audio;
rc[1] = tv;
rc[2] = smart;

for (RemoteControl r : rc) {
    r.turnOn();
    r.setVolume(5);
    r.setMute(true);
    r.setMute(false);
    r.turnOff();
    System.out.println("-----\n");
}
```

3절. 인터페이스 구현

❖ 실행 클래스 (RemoteControlEx

```
System.out.println("------\n << 다형성 결과

RemoteControl[] rc = new RemoteControl[
rc[0] = audio;
rc[1] = tv;
rc[2] = smart;

for (RemoteControl r : rc) {
    r.turnOn();
    r.setVolume(5);
    r.setMute(true);
    r.setMute(false);
    r.turnOff();
    System.out.println("-----\n");
}
```

```
<< 다형성 결과 >>

오디오를 켭니다.
현재 Audio 볼륨 : 5
Audio 무음 처리...
현재 Audio 볼륨 : 0
Audio 무음 해제~
현재 Audio 볼륨 : 5
오디오를 끕니다.
-----

TV를 켭니다.
현재 TV 볼륨 : 5
무음 처리 합니다.
현재 TV 볼륨 : 0
무음 해제 합니다.
TV를 끕니다.
-----

Smart TV를 켭니다.
현재 Smart TV 볼륨 : 5
Smart TV 무음 처리...
현재 Smart TV 볼륨 : 0
Smart TV 무음 해제~
현재 Smart TV 볼륨 : 5
Smart TV를 끕니다.
```

5절. 타입변환과 다형성

❖ 매개변수의 다형성

- 스마트폰에 블루투스 이어폰 연결해서 음악듣기
- 모든 이어폰은 재생(play)과 정지(stop) 기능 사용
- 다양한 블루투스 이어폰 사용 가능
  - 삼성 Buds
  - 애플 AirPods
  - LG TonFree

5절. 타입변환과 다형성

❖ 매개변수의 다형성

- 스마트폰에 블루투스 이어폰 연결해서 음악듣기
- 모든 이어폰은 재생(play)과 정지(stop) 기능 사용
- 다양한 블루투스 이어폰 사용 가능
  - 삼성 Buds
  - 애플 AirPods
  - LG TonFree

- 실행 결과

```
<< 스마트 폰으로 음악 재생하기 >>

연결할 이어폰 종류를 선택하세요(1.Buds 2.AirPods 3.TonFree 4.종료) >> 1

Buds 이어폰 연결
삼성 Buds 음악 재생 중...
삼성 Buds 음악을 중지합니다.
```

5절. 타입변환과 다형성

❖ 매개변수의 다형성

```
public interface EarPhone {
    void play();
    void stop();
}
```

```
public class AirPods implements EarPhone {
    public AirPods() {
        System.out.println("\n새로 구입한 Apple AirPods 연결");
    }

    @Override
    public void play() {
        System.out.println("Apple AirPods 음악 재생 중...");
    }
    @Override
    public void stop() {
        System.out.println("Apple AirPods 음악을 중지합니다.");
    }
}
```

5절. 타입변환과 다형성

❖ 매개변수의 다형성

```
public class Buds implements EarPhone{
    public Buds() {
        System.out.println("\nBuds 이어폰 연결 ");
    }

    //음악을 재생하는 표준 API
    public void play() {
        System.out.println("삼성 Buds 음악 재생 중...");
    }

    //음악을 정지하는 표준 API
    public void stop() {
        System.out.println("삼성 Buds 음악을 중지합니다.");
    }
}
```

```
public class TonFree implements EarPhone {
    public TonFree() {
        System.out.println("\nLG 블루투스 이어폰 톤프리 연결");
    }
    @Override
    public void play() {
        System.out.println("LG 톤프리 음악 재생 중...");
    }

    @Override
    public void stop() {
        System.out.println("LG 톤프리 음악을 중지합니다.");
    }

}
```

5절. 타입변환과 다형성

❖ 매개변수의 다형성

```
public class SmartPhone {
    EarPhone earphone;

    public void musicOn(EarPhone ep) {
        ep.play();
    }

    public void musicOff(EarPhone ep) {
        ep.stop();
    }
}
```

5절. 타입변환과 다형성

❖ 매개변수의 다형성

```
import java.util.Scanner;

public class Main2 {
    public static void main(String[] args) {
        System.out.println("*** 스마트 폰으로 음악 재생하기 ***");
        Scanner sc = new Scanner(System.in);
        int menu;

        SmartPhone sp = new SmartPhone();
        EarPhone   ep = null;

        while (true) {
            System.out.print("\n연결할 이어폰 종류를 선택하세요(1.Buds 2.AirPods 3.TonFree 4.종료) >> ");

            menu = sc.nextInt();
            if (menu == 4)
                break;

            switch(menu) {
            case 1:
                ep = new Buds(); break;
            case 2:
                ep = new AirPods(); break;
            case 3:
                ep = new TonFree(); break;
            }
            sp.musicOn(ep);
            sp.musicOff(ep);
        }

        System.out.println("프로그램 종료");
        sc.close();
    }
}
```