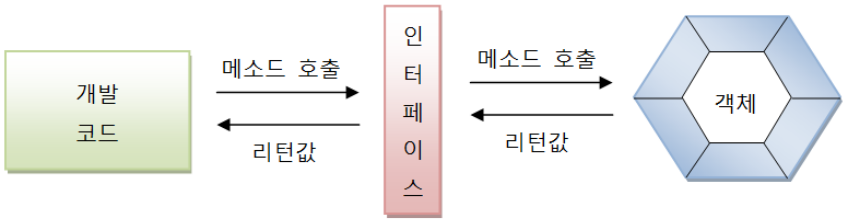
인터페이스

Contents

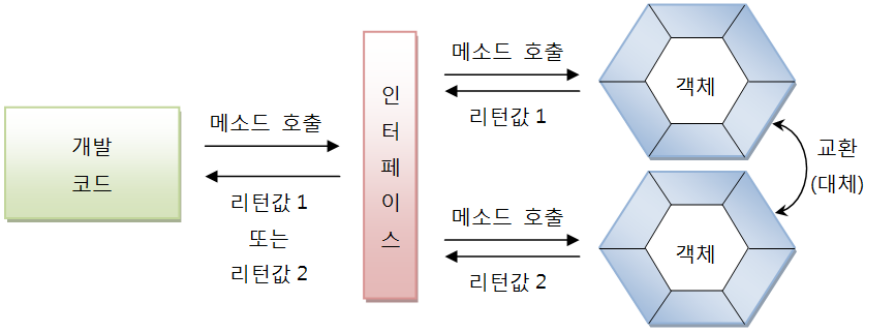
1절. 인터페이스의 역할

❖ 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

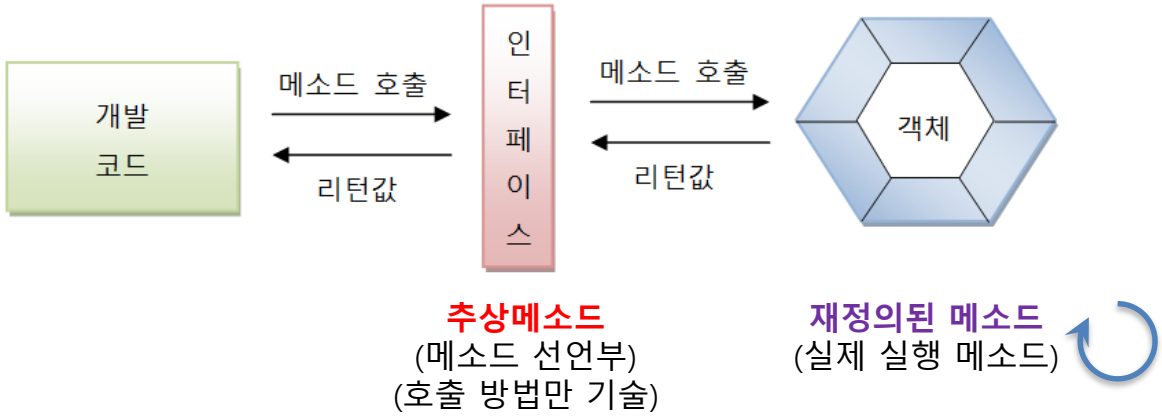
❖ 상수 필드 선언

- 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
  - 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 abstract를 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 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, ...) { ... }
```

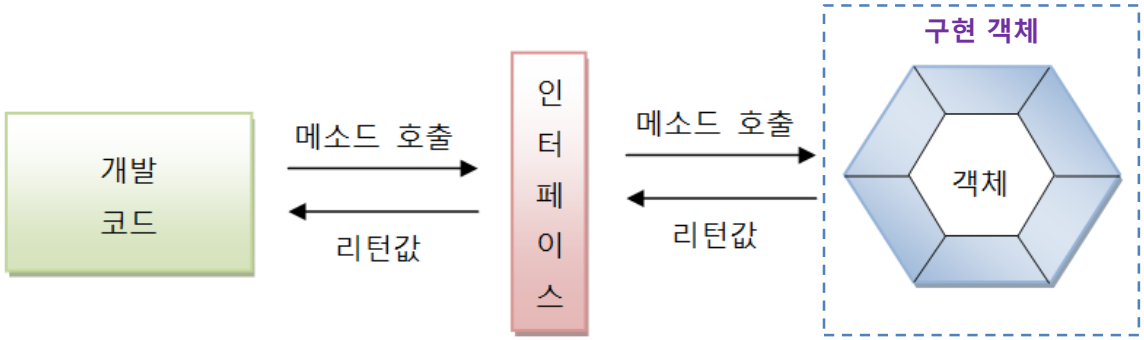
```
public interface RemoteControl {
    static void changeBattery() {
        System.out.println("건전지를 교환합니다.");
    }
}
```

- 디폴트 메소드와는 달리 객체가 없어도 인터페이스만으로 호출이 가능하다

3절. 인터페이스 구현

❖ 구현 객체와 구현 클래스

- 인터페이스의 추상 메소드 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>Volume.java)

Volume.java | Radio.java | TV.java | VolumeEx.java

```
1  package week12;
2
3  // 인터페이스 선언
4  public interface Volume {
5
6      // 인터페이스에 선언된 메소드는 실행문이 없는 추상 메소드이다.
7      // public abstract를 생략해도 컴파일 과정에서 자동으로 생성된다.
8      // 최종적으로 구현 클래스에서 정의되고 객체에서 실행된다.
9      public void volumeUp(int level);
10     public void volumeDown(int level);
11 }
```

3절. 인터페이스 구현

❖ 구현 클래스 선언 예제 (New>class>Radio.java)

Volume.java | Radio.java | TV.java

Volume 인터페이스를 구현하는 구현 클래스

```
1  package week12;
2  public class Radio implements Volume {
3      private int VolLevel;
4
5      public Radio() {
6          VolLevel = 0;
7      }
8
9      @Override
10     public void volumeUp(int level) {
11         VolLevel += level;
12         System.out.println("라디오 볼륨을 올립니다 : "+VolLevel);
13     }
14
15     @Override
16     public void volumeDown(int level) {
17         VolLevel -= level;
18         if (VolLevel < 0)
19             VolLevel = 0;
20         System.out.println("라디오 볼륨을 내립니다 : "+VolLevel);
21     }
22 }
```

3절. 인터페이스 구현

❖ 구현 클래스 선언 예제 (New>class>TV.java)

Volume.java | Radio.java | TV.java

Volume 인터페이스를 구현하는 구현 클래스

```
1  package week12;
2  public class TV implements Volume {
3      private int VolLevel;
4
5      public TV() {
6          VolLevel = 0;
7      }
8
9      @Override
10     public void volumeUp(int level) {
11         VolLevel += level;
12         System.out.println("TV 볼륨을 올립니다 : "+VolLevel);
13     }
14
15     @Override
16     public void volumeDown(int level) {
17         VolLevel -= level;
18         if (VolLevel < 0)
19             VolLevel = 0;
20         System.out.println("TV 볼륨을 내립니다 : "+VolLevel);
21     }
22 }
```

3절. 인터페이스 구현

❖ 실행 클래스 (VolumeEx.java)

Volume.java | Radio.java | TV.java | VolumeEx.java

```
1  package week12;
2
3  public class VolumeEx {
4      public static void main(String[] args) {
5          // 구현 클래스를 이용한 객체 생성
6          Radio radio = new Radio();
7          TV    tv    = new TV();
8
9          // 생성된 객체를 이용한 메소드 호출
10         radio.volumeUp(3);
11         radio.volumeDown(5);
12         tv.volumeUp(6);
13         tv.volumeDown(4);
14     }
15 }
```

3절. 인터페이스 구현

❖ 실행 클래스 (VolumeEx.java)

Volume.java | Radio.java | TV.java | VolumeEx.java

```
1  package week12;
2
3  public class VolumeEx {
4      public static void main(String[] args) {
5          // 구현 클래스를 이용한 객체 생성
6          Radio radio = new Radio();
7          TV    tv    = new TV();
8
9          // 생성된 객체를 이용한 메소드 호출
10         radio.volumeUp(3);
11         radio.volumeDown(5);
12         tv.volumeUp(6);
13         tv.volumeDown(4);
14     }
15 }
```

Console

<terminated> VolumeEx [Java Application] C:₩P

```
라디오 볼륨을 올립니다 : 3
라디오 볼륨을 내립니다 : 0
TV 볼륨을 올립니다 : 6
TV 볼륨을 내립니다 : 2
```

3절. 인터페이스 구현

❖ 인터페이스-상수 필드 (Volume.java)

Volume.java | Radio.java | TV.java | VolumeEx.java

```
1  package week12;
2
3  // 인터페이스 선언
4  public interface Volume {
5      // 인터페이스는 객체 사용 설명서이므로 런타임 시 데이터를 저장할 수 있는 필드를 선언할 수 없다.
6      // 인터페이스에서 선언 가능한 것은 상수(static final) 필드이다.
7      // 상수를 선언할 때는 반드시 초기값을 설정해야 한다.
8      // static final을 생략해도 컴파일 과정에서 자동 생성된다.
9      public int MAX_VOLUME = 20;
10     public int MIN_VOLUME = 0;
11
12     // 인터페이스에 선언된 메소드는 실행문이 없는 추상 메소드이다.
13     // public abstract를 생략해도 컴파일 과정에서 자동으로 생성된다.
14     // 최종적으로 구현 클래스에서 정의되고 객체에서 실행된다.
15     public void volumeUp(int level);
16     public void volumeDown(int level);
17 }
```

3절. 인터페이스 구현

❖ 구현 클래스-상수 필드 사용 (Radio.java)

Volume.java | Radio.java | TV.java | VolumeEx.java

```
1  package week12;
2  public class Radio implements Volume {
3      private int VolLevel;
4
5      public Radio() {
6          VolLevel = 0;
7      }
8
9      @Override
10     public void volumeUp(int level) {
11         VolLevel += level;
12         if (VolLevel > Volume.MAX_VOLUME)
13             VolLevel = Volume.MAX_VOLUME;
14         System.out.println("라디오 볼륨을 올립니다 : "+VolLevel);
15     }
16
17     @Override
18     public void volumeDown(int level) {
19         VolLevel -= level;
20         if (VolLevel < Volume.MIN_VOLUME)
21             VolLevel = Volume.MIN_VOLUME;
22         System.out.println("라디오 볼륨을 내립니다 : "+VolLevel);
23     }
24 }
```

3절. 인터페이스 구현

❖ 구현 클래스-상수 필드 사용 (TV.java)

Volume.java | Radio.java | TV.java | VolumeEx.java

```
1  package week12;
2  public class TV implements Volume {
3      private int VolLevel;
4
5      public TV() {
6          VolLevel = 0;
7      }
8
9      @Override
10     public void volumeUp(int level) {
11         VolLevel += level;
12         if (VolLevel > Volume.MAX_VOLUME)
13             VolLevel = Volume.MAX_VOLUME;
14         System.out.println("TV 볼륨을 올립니다 : "+VolLevel);
15     }
16
17     @Override
18     public void volumeDown(int level) {
19         VolLevel -= level;
20         if (VolLevel < Volume.MIN_VOLUME)
21             VolLevel = Volume.MIN_VOLUME;
22         System.out.println("TV 볼륨을 내립니다 : "+VolLevel);
23     }
24 }
```

3절. 인터페이스 구현

❖ 실행 클래스 (VolumeEx.java)

Volume.java | Radio.java | TV.java | VolumeEx.java

```
1  package week12;
2
3  public class VolumeEx {
4      public static void main(String[] args) {
5          // 구현 클래스를 이용한 객체 생성
6          Radio radio = new Radio();
7          TV    tv    = new TV();
8
9          // 생성된 객체를 이용한 메소드 호출
10         radio.volumeUp(3);
11         radio.volumeDown(5);
12         tv.volumeUp(6);
13         tv.volumeDown(4);
14
15         // 인터페이스를 이용한 메소드 호출
16         Volume[] vol = new Volume[2];
17         vol[0] = radio;
18         vol[1] = tv;
19
20         for (int i=0; i<2; i++)
21             vol[i].volumeUp(3);
22     }
23 }
```

3절. 인터페이스 구현

❖ 실행 클래스 (VolumeEx.java)

Volume.java | Radio.java | TV.java | VolumeEx.java

```
1  package week12;
2
3  public class VolumeEx {
4      public static void main(String[] args) {
5          // 구현 클래스를 이용한 객체 생성
6          Radio radio = new Radio();
7          TV    tv    = new TV();
8
9          // 생성된 객체를 이용한 메소드 호출
10         radio.volumeUp(3);
11         radio.volumeDown(5);
12         tv.volumeUp(6);
13         tv.volumeDown(4);
14
15         // 인터페이스를 이용한 메소드 호출
16         Volume[] vol = new Volume[2];
17         vol[0] = radio;
18         vol[1] = tv;
19
20         for (int i=0; i<2; i++)
21             vol[i].volumeUp(3);
22     }
23 }
```

Console

<terminated> VolumeEx2 [Java Application]

```
라디오 볼륨을 올립니다 : 3
TV 볼륨을 올립니다 : 3
```

3절. 인터페이스 구현

❖ 인터페이스-디폴트 메소드(Volume.java)

```
3  // 인터페이스 선언
4  public interface Volume {
5      // 인터페이스는 객체 사용 설명서이므로 런타임 시 데이터를 저장할 수 있는 필드를 선언할 수 없다.
6      // 인터페이스에서 선언 가능한 것은 상수(static final) 필드이다.
7      // 상수를 선언할 때는 반드시 초기값을 설정해야 한다.
8      // static final을 생략해도 컴파일 과정에서 자동 생성된다.
9      public int MAX_VOLUME = 20;
10     public int MIN_VOLUME = 0;
11
12     // 인터페이스에 선언된 메소드는 실행문이 없는 추상 메소드이다.
13     // public abstract를 생략해도 컴파일 과정에서 자동으로 생성된다.
14     // 최종적으로 구현 클래스에서 정의되고 객체에서 실행된다.
15     public void volumeUp(int level);
16     public void volumeDown(int level);
17
18     // 디폴트 메소드 선언 => 기존 구현 클래스에는 영향을 주지 않는다
19     default void setMute(boolean mute) {
20         if (mute)
21             System.out.println("무음 처리합니다.");
22         else
23             System.out.println("무음 해제합니다.");
24     }
25 }
```

3절. 인터페이스 구현

❖ 구현 클래스-디폴트 메소드 재정의(Speaker.java)

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

```
1  package week12;
2  public class Speaker implements Volume {
3      private boolean mute;
4      private int VolLevel;
5          public Speaker() {
6          VolLevel = 0;
7      }
8      @Override
9      public void volumeUp(int level) {
10         VolLevel += level;
11         System.out.println("스피커 볼륨을 올립니다 : "+VolLevel);
12     }
13     @Override
14     public void volumeDown(int level) {
15         VolLevel -= level;
16         if (VolLevel < 0)
17             VolLevel = 0;
18         System.out.println("스피커 볼륨을 내립니다 : "+VolLevel);
19     }
20     @Override
21     public void setMute(boolean mute) {      //디폴트 메소드 재정의
22         this.mute = mute;
23         if (mute)
24             System.out.println("스피커 무음 처리합니다.");
25         else
26             System.out.println("스피크 무음 해제합니다.");
27     }
28 }
```

3절. 인터페이스 구현

❖ 실행 클래스 (VolumeEx.java)

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

```
1  package week12;
2
3  public class VolumeEx {
4      public static void main(String[] args) {
5          // 구현 클래스를 이용한 객체 생성
6          Radio radio = new Radio();
7          TV    tv    = new TV();
8          // 생성된 객체를 이용한 메소드 호출
9          radio.volumeUp(3);
10         radio.volumeDown(5);
11         tv.volumeUp(6);
12         tv.volumeDown(4);
13         System.out.println("--------------------");
14
15         // 인터페이스를 이용한 메소드 호출
16         Volume[] vol = new Volume[3];
17         vol[0] = radio;
18         vol[1] = tv;
19         vol[2] = new Speaker();
20
21         for (int i=0; i<3; i++) {
22             vol[i].volumeUp(3);
23             vol[i].setMute(true);
24         }
25     }
26 }
```

3절. 인터페이스 구현

❖ 실행 클래스 (VolumeEx.java)

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

```
1  package week12;
2
3  public class VolumeEx {
4      public static void main(String[] args) {
5          // 구현 클래스를 이용한 객체 생성
6          Radio radio = new Radio();
7          TV    tv    = new TV();
8          // 생성된 객체를 이용한 메소드 호출
9          radio.volumeUp(3);
10         radio.volumeDown(5);
11         tv.volumeUp(6);
12         tv.volumeDown(4);
13         System.out.println("--------------------");
14
15         // 인터페이스를 이용한 메소드 호출
16         Volume[] vol = new Volume[3];
17         vol[0] = radio;
18         vol[1] = tv;
19         vol[2] = new Speaker();
20
21         for (int i=0; i<3; i++) {
22             vol[i].volumeUp(3);
23             vol[i].setMute(true);
24         }
25     }
26 }
```

Console

<terminated> VolumeEx [Java Application] C:₩P

```
라디오 볼륨을 올립니다 : 3
라디오 볼륨을 내립니다 : 0
TV 볼륨을 올립니다 : 6
TV 볼륨을 내립니다 : 2
--------------------
라디오 볼륨을 올립니다 : 3
무음 처리합니다.
TV 볼륨을 올립니다 : 5
무음 처리합니다.
스피커 볼륨을 올립니다 : 3
스피커 무음 처리합니다.
```

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 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 | 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 }
```

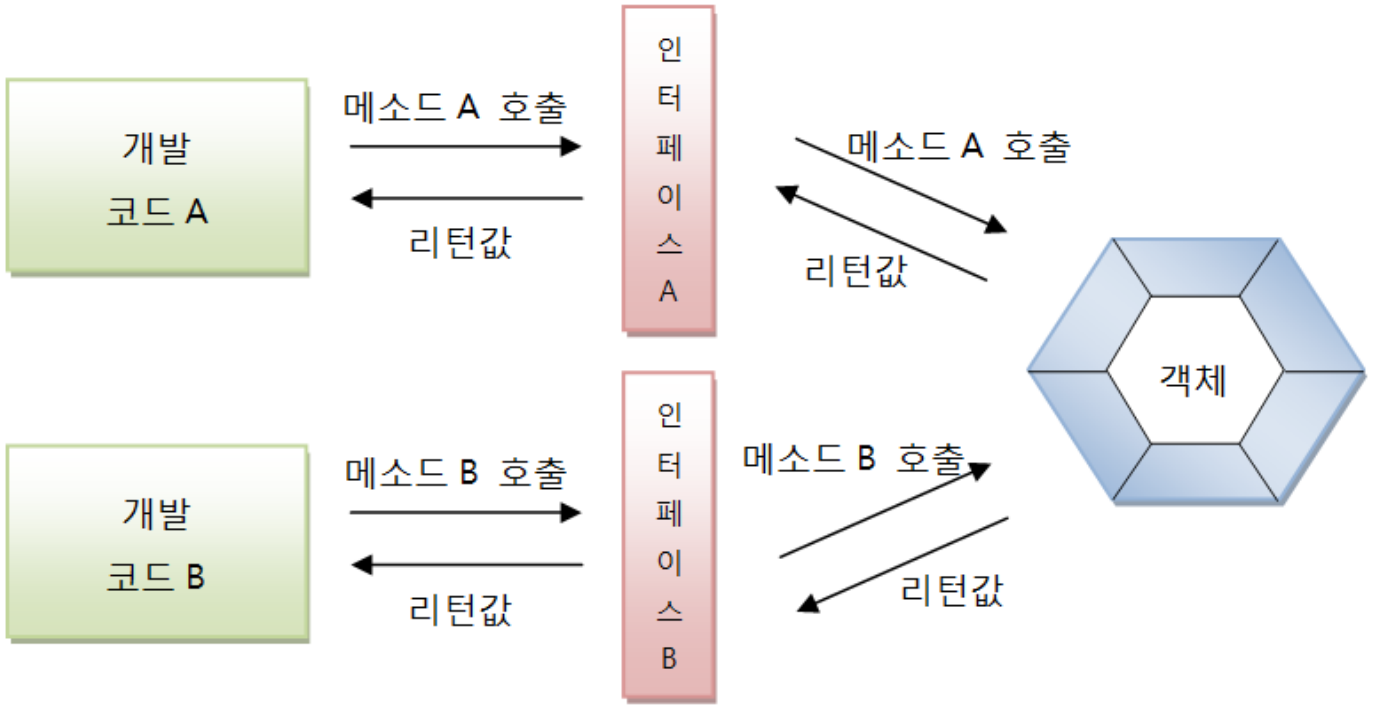
Console

<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

```
1  package week12;
2
3  public interface Searchable {
4      void search(String url);
5  }
```

Searchable.java (인터페이스)

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

```
1  package week12;
2
3  public class SmartTV implements Volume, Searchable {
4
5      @Override
6      public void search(String url) {
7          System.out.println(url + "을 검색합니다.");
8      }
9
10     @Override
11     public void volumeUp(int level) {
12         System.out.println("스마트 TV 볼륨을 올립니다 :" + level);
13     }
14
15     @Override
16     public void volumeDown(int level) {
17         System.out.println("스마트 TV 볼륨을 내립니다 :" + level);
18     }
19 }
```

SmartTV.java (클래스)

3절. 인터페이스 구현

❖ 다중 인터페이스 구현 클래스 (SmartTVEx.java)

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

```
1  package week12;
2
3  public class SmartTVEx {
4      public static void main(String[] args) {
5          SmartTV smart = new SmartTV();
6          smart.volumeUp(3);
7          smart.volumeDown(2);
8          smart.search("네이버");
9      }
10 }
11
```

3절. 인터페이스 구현

❖ 다중 인터페이스 구현 클래스 (SmartTVEx.java)

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

```
1  package week12;
2
3  public class SmartTVEx {
4      public static void main(String[] args) {
5          SmartTV smart = new SmartTV();
6          smart.volumeUp(3);
7          smart.volumeDown(2);
8          smart.search("네이버");
9      }
10 }
11
```

Console

<terminated> SmartTVEx [Java Application] C:₩

```
스마트 TV 볼륨을 올립니다 :3
스마트 TV 볼륨을 내립니다 :2
네이버을 검색합니다.
```

4절. 인터페이스 사용

❖ 인터페이스에 구현 객체를 대입하는 방법

```
인터페이스 변수;
변수 = 구현객체;
```

```
인터페이스 변수 = 구현객체;
```

```
Volume  vol;
vol = new Radio();
vol = new TV();
```

```
Volume  vol = new Radio();
```

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

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

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

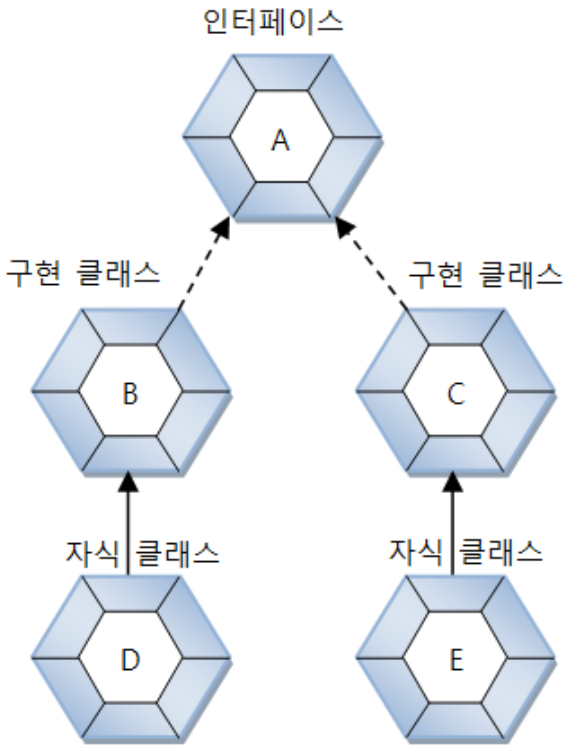
- 다형성의 효과
  - 다양한 실행 결과를 얻을 수 있음
  - 객체를 부품화시킬 수 있어 유지보수 용이 (메소드의 매개변수로 사용)

5절. 타입변환과 다형성

❖ 자동 타입 변환(Promotion)

자동 타입 변환

인터페이스 변수 = 구현객체;




```
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

[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)

- 강제 타입 변환 전 구현 클래스 타입 조사

5절. 타입변환과 다형성

❖ 매개변수의 다형성

InterfaceA.java | ClassC.java | PolyEx.java

```java
package week12;

public interface InterfaceA {
    public void methodA();
}

interface InterfaceB {
    public void methodB();
}
```

InterfaceA.java (인터페이스)

InterfaceA.java | ClassC.java | PolyEx.java

```java
package week12;
public class ClassC implements InterfaceA, InterfaceB{
    @Override
    public void methodB() {
        System.out.println("methodB");
    }
    @Override
    public void methodA() {
        System.out.println("methodA");
    }
}
```

ClassC.java (클래스)

5절. 타입변환과 다형성

❖ 매개변수의 다형성

InterfaceA.java | ClassC.java | PolyEx.java

```
1  package week12;
2
3  public class PolyEx {
4      public static void main(String[] args) {
5          ClassC     classC = new ClassC();
6          InterfaceA interA = new ClassC();    //오른쪽 구현객체가 왼쪽 인터페이스 변수에 대입
7          InterfaceB interB = new ClassC();    //이 때, 자동 타입 변환이 일어난다
8
9          classC.methodA();          // 클래스 ClassC가 정의한 모든 멤버를 호출할 수 있다
10         classC.methodB();          // 클래스 ClassC가 정의한 모든 멤버를 호출할 수 있다
11
12         interA.methodA();          // 인터페이스 InterfaceA에 정의된 메소드만 호출 가능
13         //interA.methodB();        // 자동 변환 후에는 해당 인터페이스 멤버만 호출할 수 있다
14
15         //interB.methodA();        // ClassC에 정의되어 있어도 인터페이스로 자동변환되면서 호출 불가능
16         interB.methodB();          // 인터페이스 InterfaceB에 정의된 메소드만 호출 가능
17     }
18 }
```

5절. 타입변환과 다형성

❖ 매개변수의 다형성

InterfaceA.java | ClassC.java | PolyEx.java

```
4   public static void main(String[] args) {
5       ClassC     classC = new ClassC();
6       InterfaceA interA = new ClassC();    //오른쪽 구현객체가 왼쪽 인터페이스 변수에 대입
7       InterfaceB interB = new ClassC();    //이 때, 자동 타입 변환이 일어난다
8
9       classC.methodA();          // 클래스 ClassC가 정의한 모든 멤버를 호출할 수 있다
10      classC.methodB();          // 클래스 ClassC가 정의한 모든 멤버를 호출할 수 있다
11
12      interA.methodA();          // 인터페이스 InterfaceA에 정의된 메소드만 호출 가능
13      //interA.methodB();        // 자동 변환 후에는 해당 인터페이스 멤버만 호출할 수 있다
14
15      if (interA instanceof ClassC) {      // 객체 타입 확인 후
16          ClassC castA = (ClassC)interA;   // 강제 타입 변환
17          castA.methodB();
18      }
19
20      //interB.methodA();        // ClassC에 정의되어 있어도 인터페이스로 자동변환되면서 호출 불가능
21      interB.methodB();          // 인터페이스 InterfaceB에 정의된 메소드만 호출 가능
22
23      if (interB instanceof ClassC) {
24          ClassC castB = (ClassC)interB;
25          castB.methodA();
26      }
27  }
28 }
```

6절. 인터페이스 상속

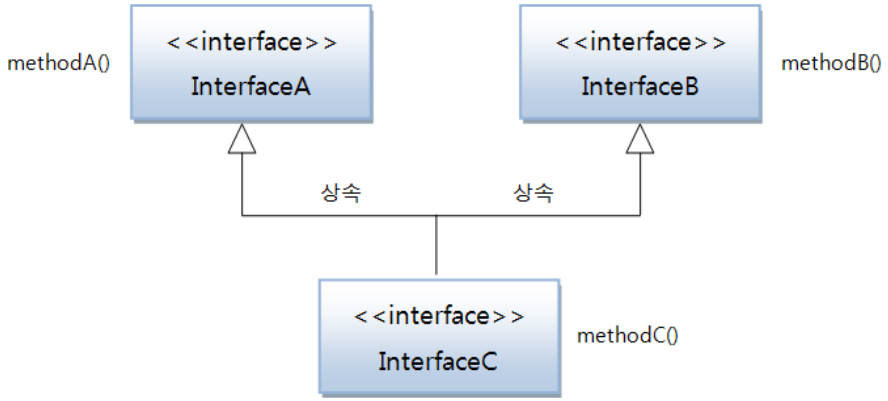
❖ 인터페이스간 상속 가능

```
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**

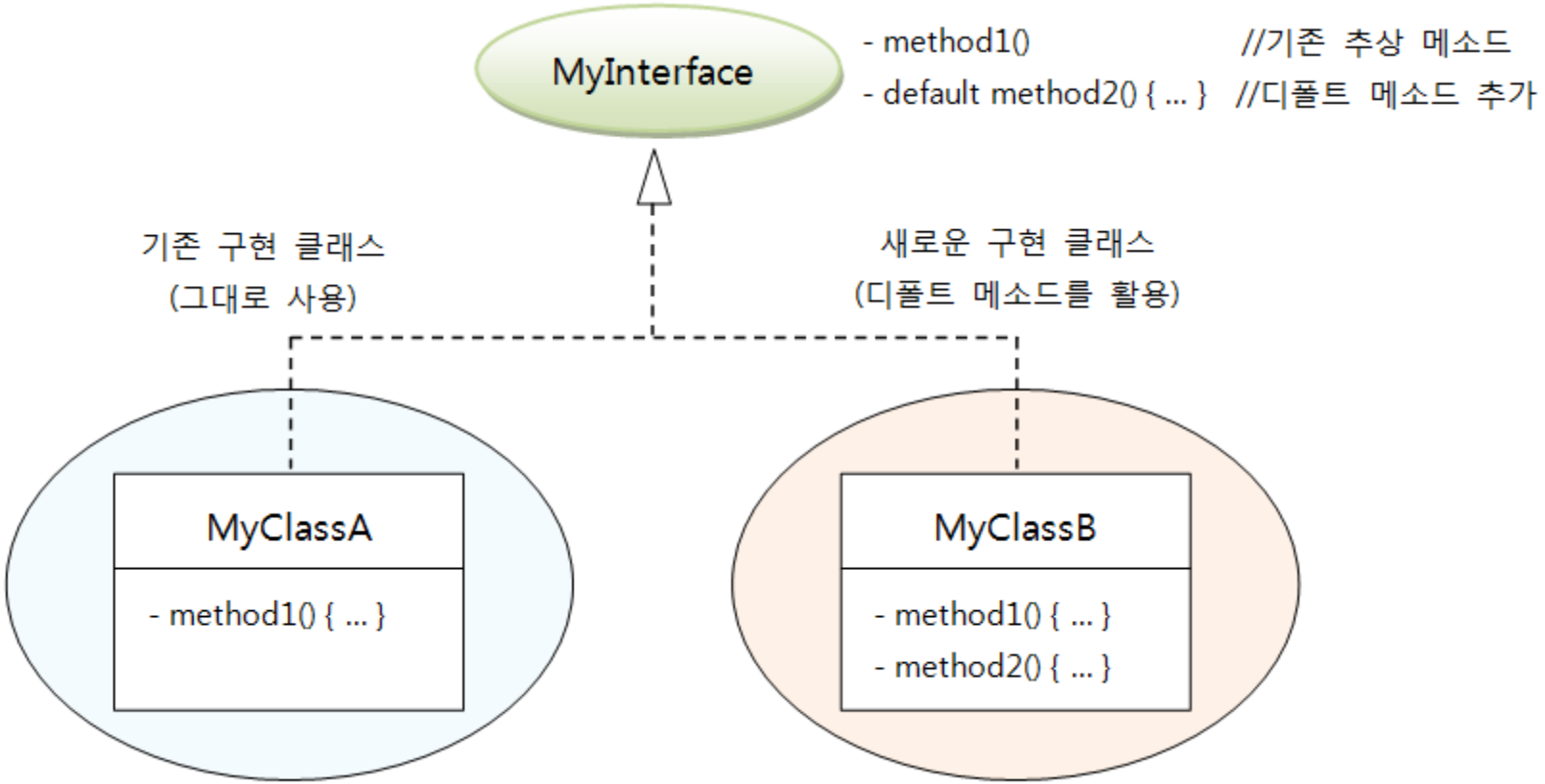
```
하위인터페이스 변수 = new 구현클래스(...);
상위인터페이스1 변수 = new 구현클래스(...);
상위인터페이스2 변수 = new 구현클래스(...);
```

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



7절. 디폴트 메소드와 인터페이스 확장

❖ 디폴트 메소드와 확장 메소드 사용하기



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

인터페이스를 이용한 도서대출 프로그램

❖ 인터페이스

Lendable.java | Book.java | BookEx.java

```
1  package week12;
2
3  public interface Lendable {
4      public byte STATE_BORROWED = 1;      //대출중
5      public byte STATE_NORMAL   = 0;      //대출 가능
6
7      void checkOut(String person, String date);
8      void checkIn();
9  }
```

인터페이스를 이용한 도서대출 프로그램

❖ 인터페이스를 구현한 클래스

Lendable.java | Book.java | BookEx.java

```
3  public class Book implements Lendable{
4
5      String bookNo;          // 청구번호
6      String title;           // 제목
7      String writer;          // 저자
8      String person;          // 대출자
9      String checkDate;       // 대출일자
10     byte   state;           // 대출 상태
11
12     public Book(String bookNo, String title, String writer) {
13         this.bookNo = bookNo;
14         this.title  = title;
15         this.writer = writer;
16     }
17
```

continued...

인터페이스를 이용한 도서대출 프로그램

❖ 인터페이스를 구현한 클래스

Lendable.java | Book.java | BookEx.java

```
17
18      @Override
19      public void checkOut(String person, String date) {
20          if (state == Lendable.STATE_BORROWED) {
21              System.out.println("이미 대출된 도서입니다.");
22              return;
23          }
24
25          this.person    = person;
26          this.checkDate = date;
27          this.state     = Lendable.STATE_BORROWED;
28
29          System.out.println("* " + title + " 도서가 대출되었습니다.");
30          System.out.println("대출인 : " + person);
31          System.out.println("대출일 : " + checkDate);
32      }
33
34      @Override
35      public void checkIn() {
36          this.person    = null;
37          this.checkDate = null;
38          this.state     = Lendable.STATE_NORMAL;
39          System.out.println("▶" + title +" 도서가 반납되었습니다.");
40      }
41  }
```

인터페이스를 이용한 도서대출 프로그램

❖ 실행 클래스

Lendable.java | Book.java | BookEx.java

```
1  package week12;
2
3  public class BookEx {
4      public static void main(String[] args) {
5          Book book = new Book("863ㅂ", "코스모스", "칼세이건");
6          printState(book);
7          book.checkOut("홍길동", "2019-05-19");
8          printState(book);
9          book.checkOut("홍길동", "2019-05-20");
10         book.checkIn();
11     }
12
13     private static void printState(Book book) {
14         if (book.state == Lendable.STATE_NORMAL) {
15             System.out.println("----------------");
16             System.out.println("제목 : " + book.title);
17             System.out.println("대출 상태 : 대출 가능");
18             System.out.println("----------------");
19         } else if (book.state == Lendable.STATE_BORROWED) {
20             System.out.println("----------------");
21             System.out.println("제목 : " + book.title);
22             System.out.println("대출 상태 : 대출중");
23             System.out.println("대출자   : " + book.person);
24             System.out.println("대출일   : " + book.checkDate);
25             System.out.println("----------------");
26         }
27     }
28 }
```

과제물

❖ 실행 클래스

BookEx 클래스를 실행한 결과창을 화면 캡쳐 후 제출
과제 제출 파일명 : 학번+이름

Lendable.java | Book.java | BookEx.java

```java
package week12;

public class BookEx {
    public static void main(String[] args) {
        Book book = new Book("863ㅂ", "코스모스", "칼세이건");
        printState(book);
        book.checkOut("홍길동", "2019-05-19");
        printState(book);
        book.checkOut("홍길동", "2019-05-20");
        book.checkIn();
    }

    private static void printState(Book book) {
        if (book.state == Lendable.STATE_NORMAL) {
            System.out.println("----------------");
            System.out.println("제목 : " + book.title);
            System.out.println("대출 상태 : 대출 가능");
            System.out.println("----------------");
        } else if (book.state == Lendable.STATE_BORROWED) {
            System.out.println("----------------");
            System.out.println("제목 : " + book.title);
            System.out.println("대출 상태 : 대출중");
            System.out.println("대출자   : " + book.person);
            System.out.println("대출일   : " + book.checkDate);
            System.out.println("----------------");
        }
    }
}
```