8장. 인터페이스

한림대학교 소프트웨어융합대학 양은샘.

8장. 인터페이스

- 안녕하세요? 여러분!
- 오늘은 자바의 인터페이스 단원을 학습 합니다.
- 이번 장에서는
  - 인터페이스의 개념과 사용법에 대해 알아보도록 하겠습니다.
  - 자바에서는 클래스, 상속, 인터페이스가 가장 중요한 부분입니다.
- 지난 시간에 학습한 내용을 리뷰한 후 학습을 시작하도록 하겠습니다.

지난 시간 Review

- 상속 개념
- 클래스 상속(extends)

- 부모 생성자 호출(super(…))
- 메소드 재정의(Override)
- final 클래스와 final 메소드

- protected 접근 제한자

- 타입변환과 다형성(polymorphism)

- 추상 클래스(abstract Class)

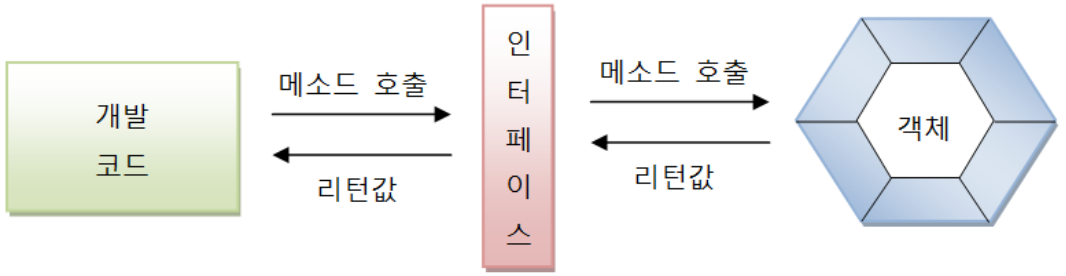
학습 목차

학습 목표

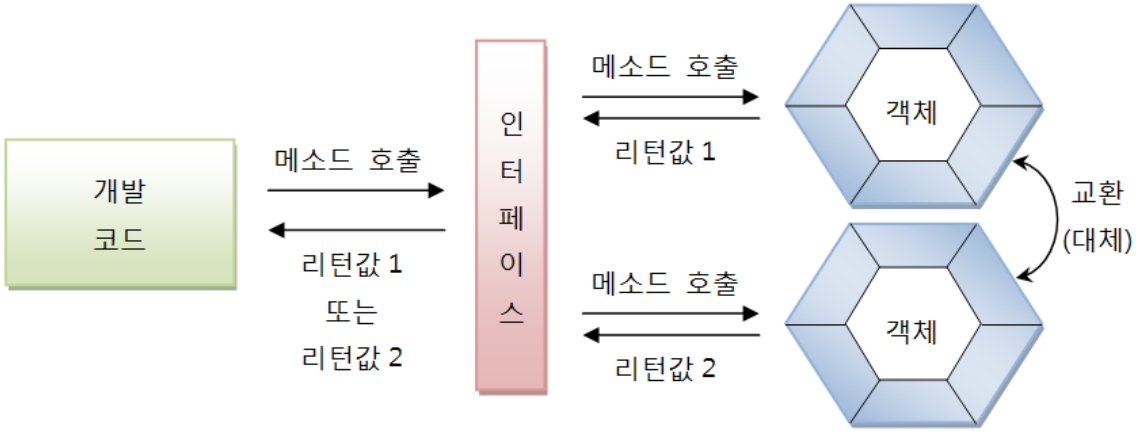
- 인터페이스의 의미와 사용법을 안다.
- 인터페이스를 이용해서 다형성을 구현할 수 있다.
- 인터페이스의 다형성을 이용해 다양한 객체를 일관된 방법으로 사용할 수 있다.

인터페이스의 역할

- 인터페이스란?
  - 개발 코드와 객체가 서로 통신하는 접점



- 인터페이스의 역할
  - 인터페이스의 메소드 호출하면 객체의 메소드가 호출된다.
  - 개발 코드가 객체에 종속되지 않게 함-> 개발 코드를 수정하지 않으면서 객체의 교환이 가능
  - 개발 코드 변경 없이 리턴값 또는 실행 내용이 다양해 질 수 있음 (다형성)



인터페이스 선언

- 인터페이스 선언

[public] interface 인터페이스명 { ... }

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

- 인터페이스 이름
  - 자바 식별자 작성 규칙에 따라 작성
  - 인터페이스 이름과 대소문자가 동일한 소스 파일 생성 : ~.java 형태

- 인터페이스는 객체를 생성할 수 없으므로 생성자를 가질 수 없다.

상수 필드 선언(constant field)

- 인터페이스는 데이터를 저장할 인스턴스 혹은 정적 필드 선언 불가

- 상수 필드만 선언 가능

  - 인터페이스에 선언된 필드는 모두 public static final
  - 자동적으로 컴파일 과정에서 붙음

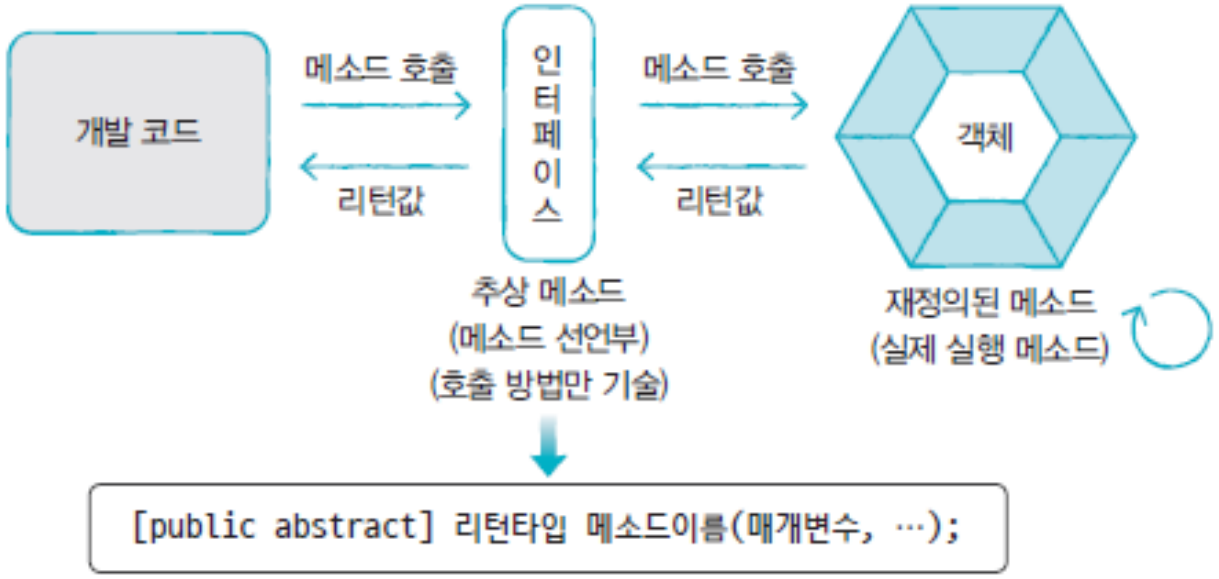
```
[public static final] 타입 상수이름 = 값;
```

  - 상수명은 대문자로 작성
  - 서로 다른 단어로 구성되어 있을 경우에는 언더스코어(_)로 연결
  - 선언과 동시에 초기값 지정
  - static { } 블록 작성 불가 - static {} 으로 초기화 불가

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

abstract 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 interface RemoteControl {
  //추상 메소드
  public void turnOn();
  public void turnOff();
  public void setVolume(int volume);
}
```

default 메소드 선언

- 자바8에서 추가된 인터페이스의 새로운 멤버
  - 실행 블록을 가지고 있는 메소드

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

- 모든 구현 객체가 가지고 있는 기본 메소드로 사용 가능
  - 필요에 따라 구현 클래스가 디폴트 메소드 재정의해 사용
- default 키워드를 반드시 붙여야 함
- 기본적으로 public 접근 제한
  - 생략하더라도 컴파일 과정에서 자동 붙음
- 인터페이스만으로는 사용 불가
  - 구현 객체가 인터페이스에 대입되어야 호출할 수 있는 인스턴스 메소드

정적 메소드 선언

- 자바8에서 추가된 인터페이스의 새로운 멤버

- 정적 메소드 사용
  - 인터페이스로 바로 호출 가능

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

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

인터페이스 구현

- 구현 (implement) 클래스
  - 인터페이스에서 정의된 추상 메소드를 재정의해서 실행 내용을 가지고 있는 클래스

```
public class 구현클래스이름 implements 인터페이스이름 {
  //인터페이스에 선언된 추상 메소드의 실체 메소드 선언
}
```

```
public class Television implements RemoteControl {
  //turnOn() 추상 메소드의 실체 메소드
  public void turnOn() {
    System.out.println("TV를 켭니다.");
  }
  //turnOff() 추상 메소드의 실체 메소드
  public void turnOff() {
    System.out.println("TV를 끕니다.");
  }
```

- 추상 메소드의 실체 메소드를 작성하는 방법
  - 클래스 선언부에 implements 키워드 추가하고 인터페이스 이름 명시
  - 메소드의 선언부가 정확히 일치해야
  - 인터페이스의 모든 추상 메소드를 재정의하는 실체 메소드 작성해야 함
  - 일부만 재정의할 경우, 추상 클래스로 선언 + abstract 키워드 붙임

익명 구현 객체

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

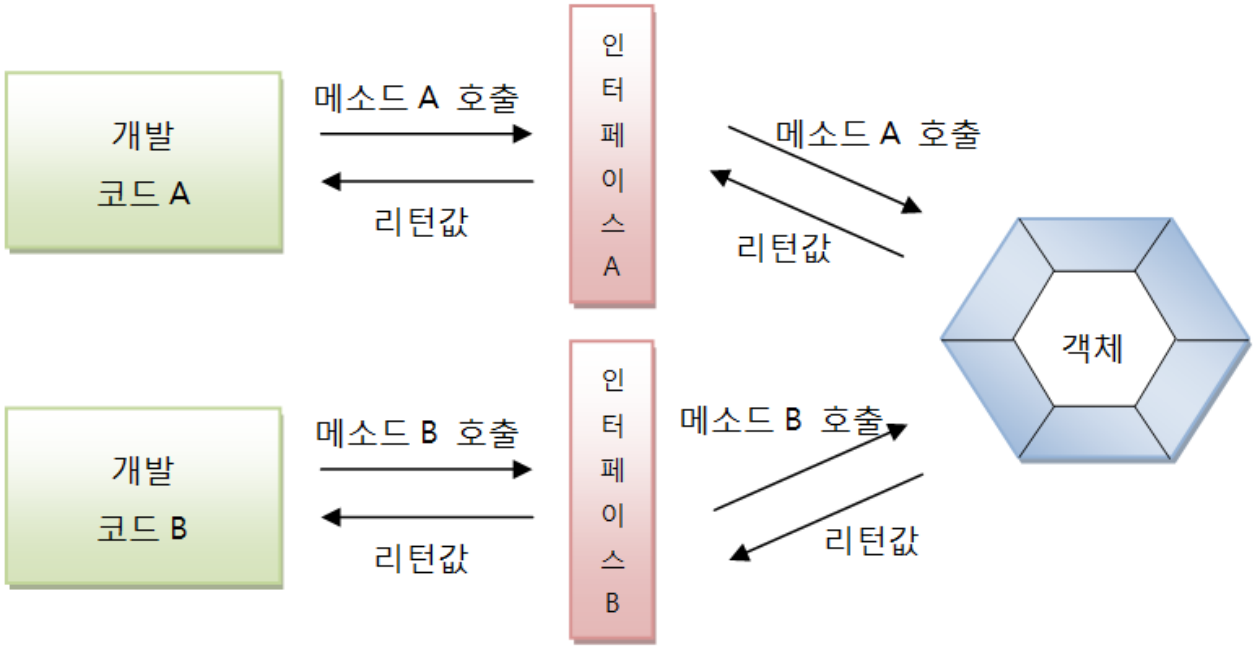
```
인터페이스 변수 = new 인터페이스() {
  //인터페이스에 선언된 추상 메소드의 실체 메소드 선언
};
```

- 인터페이스의 추상 메소드들을 모두 재정의하는 실체 메소드가 있어야 함.
- 추가적으로 필드와 메소드 선언 가능하나 익명 객체 안에서만 사용

- 인터페이스 변수로 접근 불가

다중 인터페이스 구현 클래스

❖ 객체는 다수의 인터페이스 타입으로 사용 가능




```
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
}
```

인터페이스 사용

❖ 인터페이스는 필드, 매개 변수, 로컬 변수로 선언 가능

```
public class MyClass {
  //필드
① RemoteControl rc = new Television();

  //생성자
② MyClass( RemoteControl rc ) {
    this.rc = rc;
  }

  //메소드
  void methodA() {
    //로컬 변수
③ RemoteControl rc = new Audio();
  }

④ void methodB( RemoteControl rc ) { … }
}
```

생성자의 매개값으로 구현 객체 대입

```
MyClass mc = new MyClass(new Television());
```

생성자의 매개값으로 구현 객체 대입

```
mc.methodB(new Audio());
```

인터페이스 사용 예

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

타입변환과 다형성(Polymorphism)

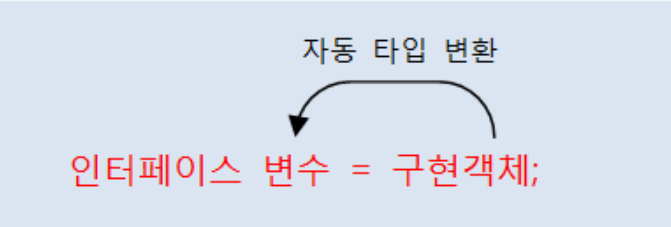
- 다형성
  - 하나의 타입에 여러 가지 객체를 대입해 다양한 실행 결과를 얻는 것

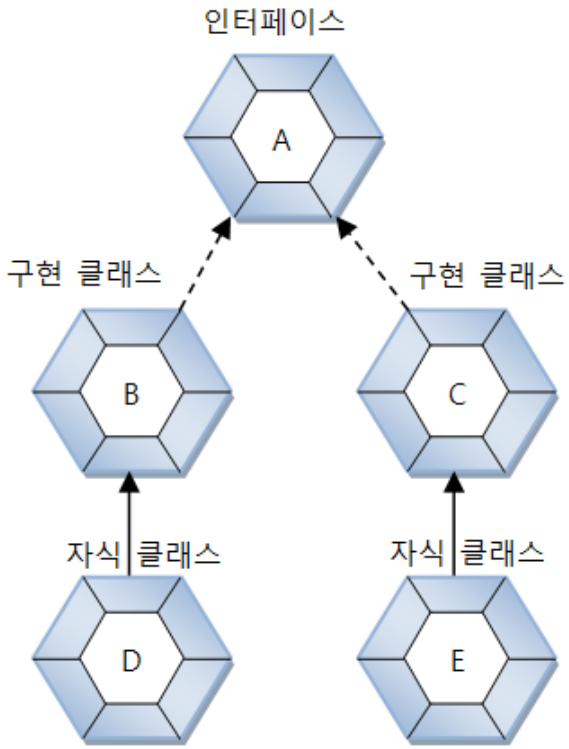
- 다형성을 구현하는 기술
  - 상속 또는 인터페이스의 자동 타입 변환(Promotion)
  - 오버라이딩(Overriding)

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

자동 타입 변환(Promotion)

❖ 구현 객체와 자식 객체는 인터페이스 타입으로 자동 타입 변환 가능






```
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)
```

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();
```

인터페이스 배열로 구현한 객체 관리

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

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

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

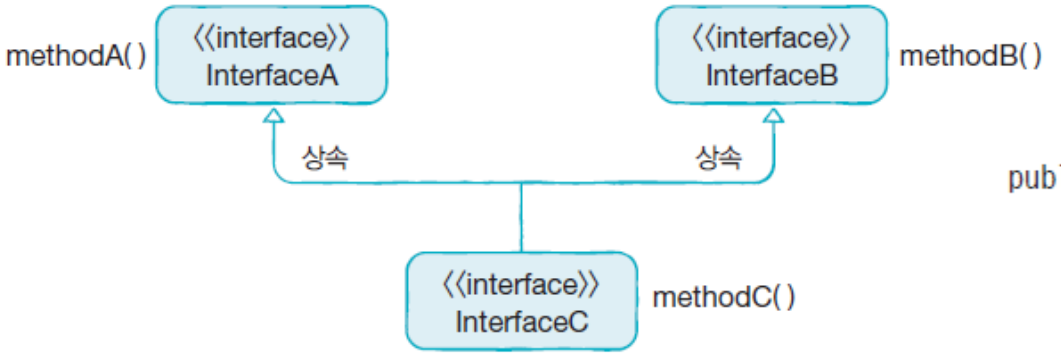
```
tires[1] = new KumhoTire();
```

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

인터페이스 상속

❖ 인터페이스는 다중 상속을 할 수 있다.

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




```
public interface InterfaceC extends InterfaceA, InterfaceB {
```

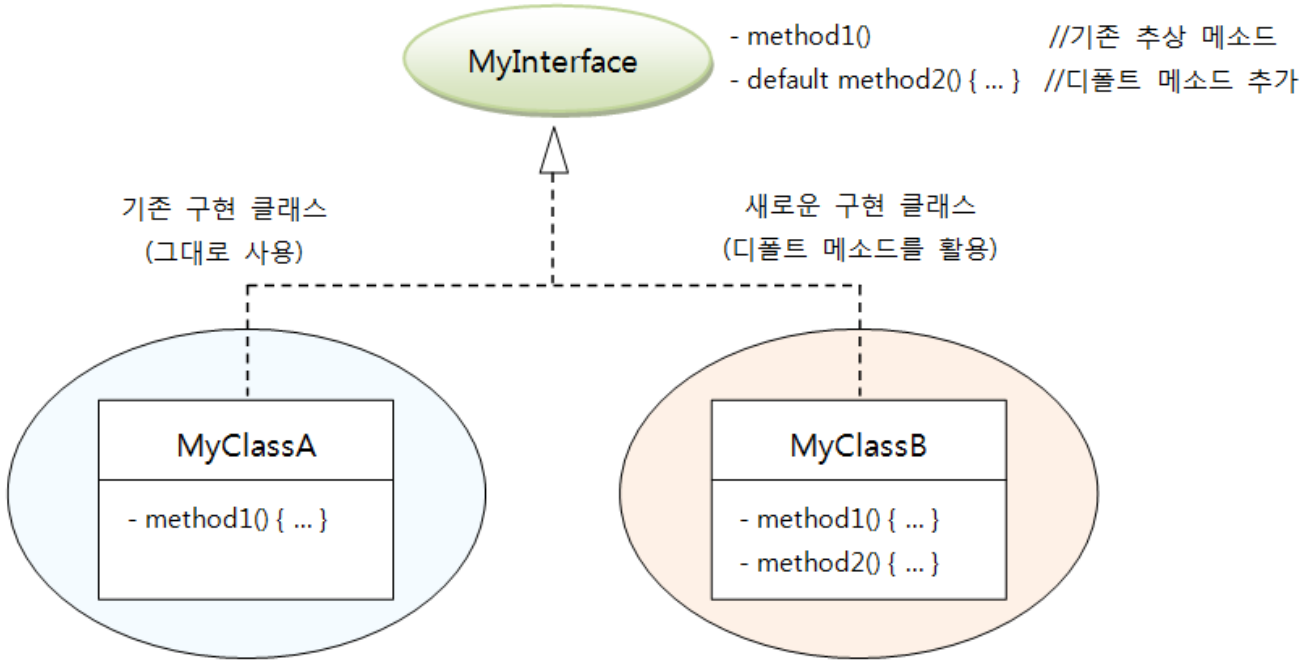
❖ 인터페이스 자동 타입 변환

```
public class ImplementationC implements InterfaceC {
        ImplementationC impl = new ImplementationC();
        InterfaceA ia = impl;
        InterfaceB ib = impl;
        InterfaceC ic = impl;
```

❖ 하위 인터페이스의 구현 클래스는 상위 인터페이스의 모든 추상 메소드를 재정의해야 함

디폴트 메소드와 인터페이스 확장

- 디폴트 메소드가 있는 인터페이스를 상속 했을 경우 자식 인터페이스에서 활용하는 방법
  - 부모 인터페이스의 디폴트 메소드를 단순히 상속만 받을 수도 있음
  - 디폴트 메소드를 재정의(Override)해서 실행 내용 변경도 가능
  - 디폴트 메소드를 추상 메소드로 재선언도 가능



인터페이스 중요 소스

- java_08장_인터페이스_중요소스.pdf

적용 확인 학습 & 응용 프로그래밍

- 다음 파일에 있는 문제들의 해답을 스스로 작성 해 보신 후 개념 & 적용 확인 학습 영상을 학습 하시기 바랍니다.
  - java_08장_인터페이스_ex.pdf

- 퀴즈와 과제가 출제되었습니다.
  - 응용 프로그래밍 영상을 학습하신 후 과제와 퀴즈를 수행 하시기 바랍니다.

학습 정리

- 인터페이스
  - 객체의 사용 방법 정의한 것
  - 인터페이스의 상수 필드 : 인터페이스의 필드는 기본적으로 public static final 특성 가짐
  - 인터페이스의 메소드는 public abstrac이 생략된 메소드 선언부만 있는 추상 메소드이다.
  - 인터페이스는 다중 상속 허용한다.

- implments
  - 구현 클래스에서 어떤 인터페이스를 사용 가능한지 기술하기 위해 사용한다.
  - 인터페이스 사용 : 클래스 선언 시 필드, 매개 변수, 로컬 변수로 선언 가능. 구현 객체를 대입.
  - 자동 타입 변환 : 구현 객체는 인터페이스 변수로 자동 타입 변환된다.

- 다형성
  - 인터페이스도 재정의와 타입 변환 기능을 제공하므로 다형성을 구현할 수 있다.
  - 강제 타입 변환 : 인터페이스에 대입된 구현 객체를 다시 원래 타입으로 변환하는 것을 말한다.
  - instanceof : 객체가 어떤 타입인지 조사할 때 사용한다. 강제 타입 변환 전에 사용.

Q & A

- "인터페이스"에 대한 학습이 모두 끝났습니다.

- 모든 내용을 이해 하셨나요?
- 아직 이해가 안되는 내용이 있다면 다시 한번 복습하시기 바랍니다.
- 질문은 한림 SmartLEAD 쪽지 또는 e-mail 또는 전화상담을 이용하시기 바랍니다.

- 퀴즈와 과제가 출제되었습니다. 마감시간에 늦지 않도록 주의해 주세요.

- 수고하셨습니다.^^