01.19 木 _ Interface 1

Interface

두 개의 객체의 정보를 주고 받을 수 있는 **창구** , 접점

> **Interface 는 class 와 同級 이다. ⇒ 정의를 해야 한다**

```
<기본>
interface  이름  {
  java 8 부터 이 곳에 들어올 수 있는 4가지
}
```

```
<상속 가능>
interface  extends interface이름 (+a 다중상속 가능) {
  java 8 부터 이 곳에 들어올 수 있는 4가지
}
```

interface { }안에 4 개가 올 수 있다. (3번 , 4번)

1. public **static** **final** 자료형 상수명 = 초기값 ; **class** 상수

```
interface OOO (extends interface) {
  public static final 자료형 상수명 = 초기값; // 클래스 상수가 올 수 있다.
}
```

2. public **abstract** 반환형 메서드 이름 (매개변수) ; 구현부가 없다 xx

```
interface OOO (extends interface) {
  public abstract 반환형 메서드명(매개변수); // 추상 메사드를 가져 올 수 있다.
}
```

---

java 1.8부터 적용 (기존 문제점을 보안 하기 위해 나옴)

3. public **default** 반환형 메서드 이름 (매개변수) { } **default** 메서드

```
interface OOO (extends interface) {
  public default 반환형 메서드명(매개변수) { }
}
```

4. public **static** `반환형` `메서드 이름` (`매개변수`) { } **class** 메서드

```
interface OOO (extends interface) {
  public static 반환형 메서드명(매개변수) { }
}
```

⭐ interface는 `추상 메서드`를 주로 사용한다 (⭐interface에서 가장 중요한 요소⭐)
⇒ `추상 메서드`는 구현부를 갖고 있지 않다

implements : 구현하다

❓ 왜 implements(구현하다) 라는 뜻을 사용했나?

interface의 주 속성은 `추상 메서드` , `추상 메서드` 는 구현부를 갖고 있지 않다
⇒ 때문에 구현하기 위해 implements를 사용한다

`추상 메서드`의 특징 ⇒ 메서드의 선언부만 정의해 준다

1. 구현부가 없다
2. `추상 메서드`가 들어가면 `추상 class` 가 된다
3. `추상 class`를 상속 받게 되면 부모로부터 물려받은 `추상 메서드`를 반드시 구현해야 한다

⭐interface 는 class 에 매달려서 사용된다
⭐interface 는 class와 다르게 다중 상속이 가능하다
⭐ interface 는 `추상 메서드`가 주요 속성이다 ⭐
⭐ interface 는 생성자가 없어서 인스턴스 생성 불가 ⭐

```
package test;

interface Engine {
```

```
  public abstract void starEngine();
  public abstract void stopEngine();
}

// implements : 구현하다
class Car implements Engine {       ☆ Car(class)에  Engine(interface)을 끼워넣은 상태 ☆

    public void starEngine() {      Engine에 있던 추상 메서드가 Car 달라 붙는다
      System.out.println("엔진 스타트");
    }

    public void stopEngine() {
      System.out.println("엔진 stop");
    }
}

public class TEST_1 {

  public static void main(String[] args) {

    Car obj1= new Car();

    obj1.startEngine();
    obj1.stopEngine();

  }

}
```

에러

: **Car(class)에 Engine(interface)을 붙인다. Engine에 있던 `추상 메서드`가 Car로 들어온다.**
⇒ Car는 `추상 메서드`를 갖게 된다. (starEngine( ) , stopEngine( ) **)**
⇒ Car 의 `추상 메서드`를 구현하지 않았기 때문에 에러!!

```
package test;

interface Engine {
  public abstract void starEngine();
  public abstract void stopEngine();
}

class Car implements Engine{
          내용 없음
}

public class TEST_1 {
  public static void main(String[] args) {

    Car obj1= new Car();

    obj1.startEngine();
    obj1.stopEngine();
  }
}
```

🔽

abstract 를 사용해 자식에게 넘기기면 에러 발생 X

```
package test;

interface Engine {
  public abstract void starEngine();
  public abstract void stopEngine();
}

abstract class Car implements Engine{   // 추상메서드는 구현부가 없어서 implements 를 사용
//  public void starEngine() {
//    System.out.println("엔진 스타트");
//  }
//
//  public void stopEngine() {
//    System.out.println("엔진 stop");
//  }
}

public class TEST_1 {
  public static void main(String[] args) {

    Car obj1= new Car();

    obj1.startEngine();
    obj1.stopEngine();
  }
}
```

⭐⭐⭐interface에서 정의된 `추상 메서드`들을 interface가 붙은 class 에서 구현해 사용한다⭐⭐⭐

❓ interface

interface는 서로 다른 연관성이 없는 객체와 객체를 붙여주기 위한(interface로 감싼다) **Braket** 이다

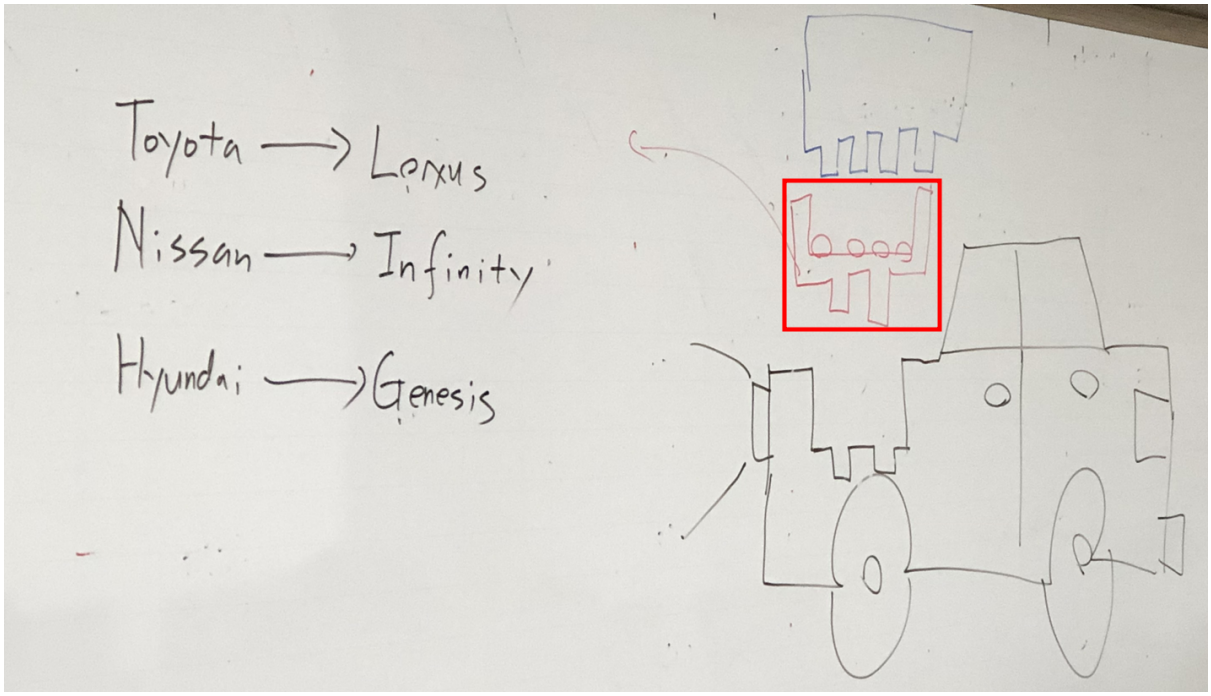
⇒ interface 는 **Braket** 이다 !

❓ interface를 class에 갖다 붙이면

interface 를 하나 정의하는데 그 안에 `추상 메서드`가 있다. 이 interface를 class에 갖다 붙인다.

그럼 이 class는 interface 의 `추상 메서드`들을 흡수해 버린다.

그럼 이 class 는 자신이 `추상 클래스`가 되어 버린다. ⇒ interface가 붙으면 그 interface에 있는 `추상메서드`들을 반드시 구현해야 한다.



예제 1

```
package level_a;

interface Engine {
  public abstract void startEngine();
  public abstract void stopEngine();
}

class Car implements Engine {
  public void startEngine() {
    System.out.println("카 엔진 스타트");
  }
  public void stopEngine() {
    System.out.println("카 엔진 스톱");
  }
}

class Bike implements Engine {
  public void startEngine() {
    System.out.println("바이크 엔진 스타트");
  }
  public void stopEngine() {
    System.out.println("바이크 엔진 스톱");
  }
}


public class A {
  public static void main(String[] args) {
    Engine obj1 = new Car();
    Engine obj2 = new Bike();
    obj1.startEngine();
    obj1.stopEngine();
    obj2.startEngine();
    obj2.stopEngine();
  }
}
```

예제 2

```
package level_a;

// 중간 매개체
interface EngineBracket {
  public abstract void startEngine();
  public abstract void stopEngine();
}

// EngineBracket 을 Benz에도 준다.
class BenzEngine implements EngineBracket{ //

  // EngineBracket 의 내용을 Benz化 시킨다
  public void startEngine() { System.out.println("벤츠 엔진 스타트"); }
  // EngineBracket 의 내용을 Benz化 시킨다
  public void stopEngine() { System.out.println("벤츠 엔진 스톱"); }
}

class G90 {
  EngineBracket myEngine;  // 참조변수 공간을 선언만 한 것
  G90(EngineBracket argEngine) {
    myEngine = argEngine;
  }

  void start() { myEngine.startEngine();}
  void stop() { myEngine.stopEngine(); }
}


public class MyFoo {
  public static void main(String args[]) {

   //g90에 BenzEngine 을 집어 넣는다
    G90 myCar = new G90( new BenzEngine() );

    myCar.start();
    myCar.stop();
  }
}
```