01.20 金 _ Interface 2

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)

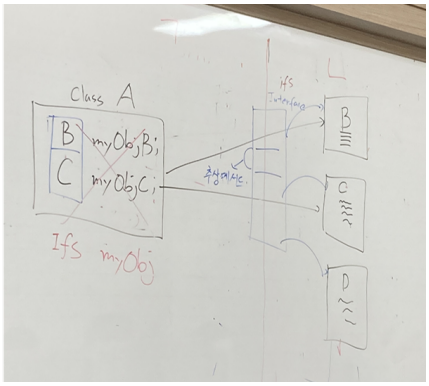
3. public default 반환형 메서드 이름 (매개변수) { } default 메서드

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

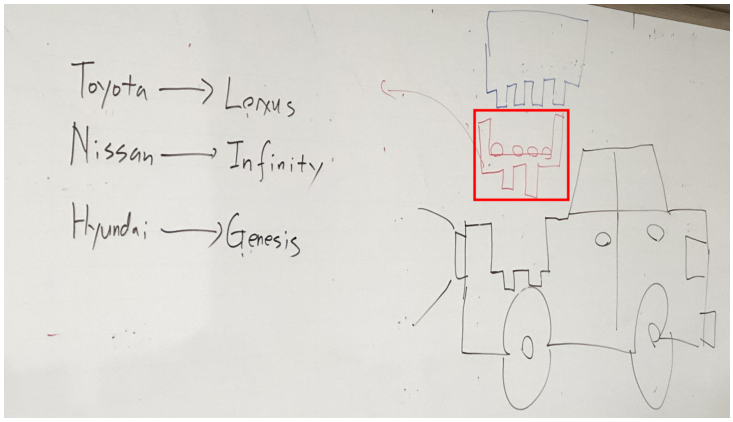
4. public static 반환형 메서드 이름 (매개변수) { } class 메서드

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

⭐ interface는 추상 메서드를 주로 사용한다 (⭐interface에서 가장 중요한 요소⭐)
⇒ 추상 메서드는 구현부를 갖고 있지 않다



interface ifs { }



🟥 : **interface**

⭐ **interface** 도 다형성이 적용 가능하다

⭐ **ifs** 로 **interface** 가 적용된 모든 객체를 가르킬 수 있다

⭐ 잘 정형화된 `추상 메서드`들을 통해서 우리가 원하는 기능만 호출해 사용하면 된다.

⭐ A 클래스는 ifs 우측 부분의 class들을 알 필요 없다.

"ㄱ" 이 **interface** 를 만든다.

~기능들을 제공하는 객체가 필요하다. 대신 그 객체는 너거가 알아서 만들어라

내가 필요한 기능들을 추상화 메서드로 추상화 시켜서 정의한 interface를 제공할 테니 너거는 너희들의 클래스를 만들어서 주면되는데 그 class에 **interface 를 끼워라.**

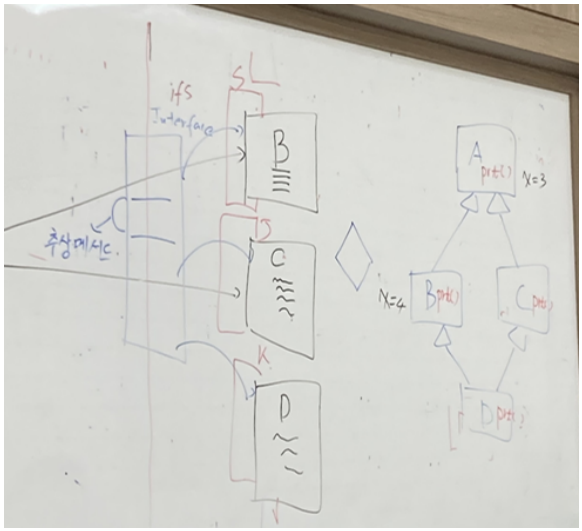
그러면 나는 너거가 만든 객체를 **interface** 형으로 다형성을 적용해서 관리하고, 그 **interface**를 내가 만들었으니 추상메소드를 호출해 이용 할테니 요구한대로 추상메서드를 잘 구현해서 내한테 줘라

interface 의 다중 상속

⭐ interface 도 상속을 제공한다

> ❗ class 는 **interface**와 상속되지 않는다!

interface 에서 다중 상속인 경우 ?



- D 에서 `prt( )`를 Overriding할 경우 B or C 어디서 Overriding 해야 하는지 알 수 없다.
  ⇒ ⚠️ ***Diamond Problem***이 발생한다 ⚠️ (모호성 문제 발생) // 다중 상속에서 항상 발생
- 다중 상속을 받으면 모호성에 대한 유효성 검사를 해야 한다 ❗
- 이런 문제로 Java에서는 다중 상속 개념을 없앴다

> **class** ⇒ **단일** 상속

> **interface** ⇒ **다중** 상속

⚠️ 상수는 *Diamond Problem*이 일어난다 ⚠️

⇒ 반대로 상수 같은 경우 값을 가진다 (모호성 문제 발생)

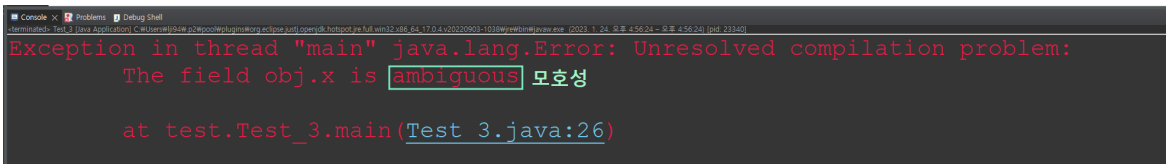
ex) x= 3   x =4 일 때 변수 이름이 같기 때문에 여기서 모호성의 문제가 발생한다.
이 같은 경우에 변수 명을 직접 바꿔 줘야 한다.

static 상수

> 모호성의 문제

```
package test;
// 다이아 몬드 프라블름
interface A {
  static final int x = 3;
}

interface B extends A {
  static final int x = 4;
}
interface C extends  A {

}
interface D extends  B , C { //  다중 상속을 하면 , 를 사용해 구현할 수 있다. => 부모 여러 개

}
```

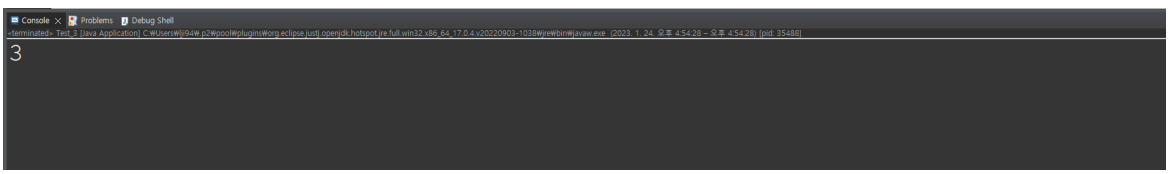
```
class Betty implements D { // 다이아몬드 형태

}

public class Test_1 {
  public static void main(String[] args) {

    Betty obj = new Betty();
    System.out.println(obj.x);
  }
}
```



🔽

변수 이름을 변경 x → y

```
package test;
// 다이아 몬드 프라블름
interface A {
  static final int x = 3;
}

interface B extends A {
  static final int y = 4;       ※ 변수명을 변경하면 된다   x -> y ※
}
interface C extends  A {

}
interface D extends  B,C { // 다이아몬드 형태

}
class Betty implements D { // 다이아몬드 형태

}

public class Test_1 {
  public static void main(String[] args) {

    Betty obj = new Betty();
    System.out.println(obj.x);
  }
}
```



⭐ 추상 메서드는 *Diamond Problem*이 일어나지 않는다 ⭐

⇒ 선언부만 있고 구현부 (알고리즘)가 없다❗❗
(호출해서 뭐가 넘어 와야 하는데 그게 없다, 알고리즘이 없다)
구현부 안에서 뭐가 일어나야 내용이 실행되는 것인데 구현부가 아예 없어서
모호성 문제가 일어나지 않는다.

※ 뭔가 틀리다를 이야기 하려면 그 안의 알고리즘이 있는 것이 전제 조건 ※

추상 메서드

> 모호성의 문제가 발생하지 않음!!

```
package test;

// 다이아 몬드 프라블름
interface A {
  public abstract void prtX();
}

interface B extends A {
  public abstract void prtX();
}

interface C extends A {
  public abstract void prtX();
}

interface D extends B, C { // 다이아몬드 형태
  public abstract void prtX();
}

class Betty implements D { // 다이아몬드 형태
  public void prtX() {
    System.out.println();
  }
}

public class Test_1 {

  public static void main(String[] args) {

    Betty obj = new Betty();
    obj.prtX();
    }
}
```

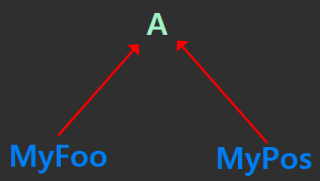
⭐ 추상 메서드는 구현부가 없이 때문에 서로 다르다는 정의를 내릴 수 가 없다.

메서드가 다르다는 것은 메서드 안의 구현부 (알고리즘)가 다르다는 것
⇒ 이 자체가 `추상 메서드`에는 없다
→ `추상 메서드`에서는 ***Diamond Problem 가 일어나지 않는다 !***

Diamond Problem

```java
package test;


interface A {
  public static final int x = 3;

  public abstract void prtX();

  public abstract void prtO();
}

class MyFoo implements A { // 다이아몬드 형태
  public void prtX() {
    System.out.println("prtX()");
  }
}

class MyPos implements A { // 다이아몬드 형태
  public void prtX() {
    System.out.println("prtX()");
  }
}

public class Test_1 {

  public static void main(String[] args) {

    A obj = new MyFoo();
    obj.prtX();
  }
}
```

```java
// Diamond Problem
interface A {
    public static final int x = 3;

    public abstract void prtX();

    public abstract void prtO();
}

class MyFoo implements A { // 다이아몬드 형태
    public void prtX() {
        System.out.println("prtX()");
    }
}

class MyPos implements A { // 다이아몬드 형태
    public void prtX() {
        System.out.println("prtX()");
    }
}
```



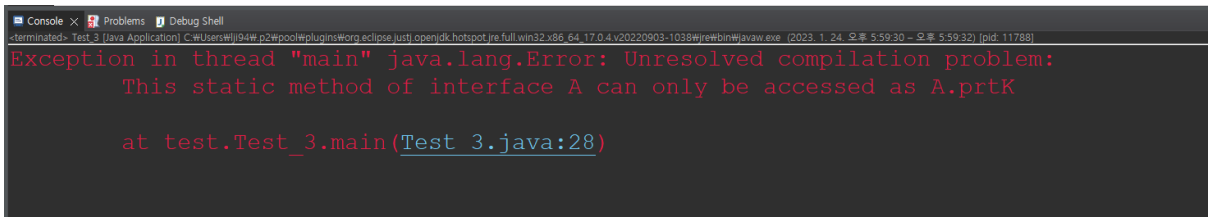
인터페이스의 상속에서도 visibility (가시性)이 존재한다.

```
package test;


interface A {
  public abstract void prtX();
}

interface B extends A {
  public abstract void prtX();
}

class Myfoo implements B { // 다이아몬드 형태
  public void prtX() {
    System.out.println("B 의 prtX()를 받아옴 : 가시성");
  }
}

public class Test_1 {
  public static void main(String[] args) {

  " A 도 가능 ( B 의 부모)"
    👇
    B obj = new Myfoo(); //  다형성 발생  ( interface  의 부모형으로 )
    obj.prtX();
    }
}
```

default 메소드 🔶 static 메소드

```
package test;

interface A {
  public static final int x = 3 ;
  public  abstract void prtX();

//Default 메소드
  public  default void prtZ() {System.out.println("Z");}
//Static  메소드
  public  static void prtK() {System.out.println("K");}

}


class Betty implements A { // 다이아몬드 형태
  public void prtX() {
    System.out.println("prtX()");
  }
}

public class Test_1 {
  public static void main(String[] args) {

    A obj = new Betty();
    obj.prtX();
    obj.prtZ();
    obj.prtK(); // Error
  }
}
```

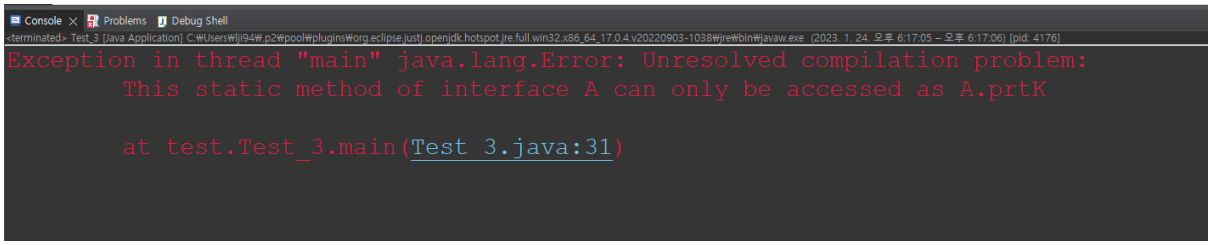
```java
public class Test_3 {

    public static void main(String[] args) {
        A obj = new Betty();

        obj.prtX();
        obj.prtZ();
        obj.prtK();
    }
}
```



interface 안의 static 메소드는 유틸리티 역할을 하기 때문에 외부에서 못 쓴다.

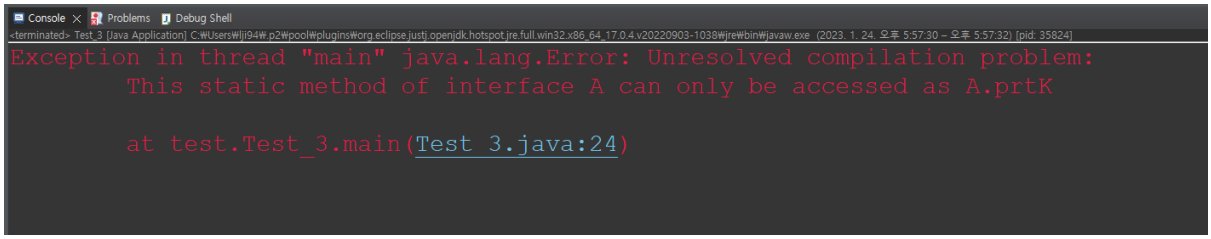
```java
package l_1;

interface A {
  public static final int x = 3;
  public abstract void prtX();

  public default void prtO() {
    prtK();
  }

  // Default method
  //public default void prtZ() { System.out.println("Z"); }

  // Static method
  public static void prtK()  {
    System.out.println("K");
  }
}

class MyFoo implements A {
  public void prtX() {
    System.out.println("prtX()");
  }
}

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

    A obj = new MyFoo();

    obj.prtK();  // Error
    A.prtK();    // static 이라서 이거는 가능
                  => class 인지 interface 인지 구분이 모호해진다
  }
}
```

```java
public class Test_3 {
    public static void main(String[] args) {

        A obj = new MyFoo();

        obj.prtK();   // Error
        A.prtK();     // static 이라서 이거는 가능
    }
}
```



기존**interface** 의 하위호환성을 가져다 주기 위해서 **default 메서드**를 사용한다

```java
package l_1;

interface A {
  public static final int x = 3;

//Default 메소드
  public default void prtZ() { System.out.println("z"); }

//Static  메소드
  public static void prtK() { System.out.println("K"); }
}

interface B extends A {
  public abstract void prtX();
}

class Foo implements B {
  public void prtX() { System.out.println("A");}
}

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

    A obj = new Foo();
    obj.prtZ(); // z
    obj.prtK(); // Error

  }
}
```


Console
Problems
Debug Shell
<terminated> Test_3 [Java Application] C:₩Users₩ji94₩.p2₩pool₩plugins₩org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.4.v20220903-1038₩jre₩bin₩javaw.exe (2023. 1. 24. 오후 5:57:30 – 오후 5:57:32) [pid: 35824]
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
This static method of interface A can only be accessed as A.prtK
at test.Test_3.main(Test_3.java:24)