JAVA 추상클래스 & 인터페이스

Prof. Youngchul Jung

주문식교육의 산실
영진전문대학

추상클래스(Abstract class)란?

- 클래스가 설계도라면 추상클래스는 '미완성 설계도'
- 추상메서드(미완성 메서드)를 포함하고 있는 클래스
  - 추상메서드 : 선언부만 있고 구현부(몸통, body)가 없는 메서드
- 일반메서드가 추상메서드를 호출할 수 있다.
- 호출할 때 필요한 건 선언부
- 완성된 설계도가 아니므로 인스턴스를 생성할 수 없다.

```
abstract class Player {
    int currentPos;          // 현재 Play되고 있는 위치를 저장하기 위한 변수

    Player() {               // 추상클래스도 생성자가 있어야 한다.
        currentPos = 0;
    }

    abstract void play(int pos);    // 추상메서드
    abstract void stop();           // 추상메서드

    void play() {
        play(currentPos);           // 추상메서드를 사용할 수 있다.
    }
    ...
}
```

추상메서드(Abstract method)란?

- 선언부만 있고 구현부(몸통, body)가 없는 메서드

```
/* 주석을 통해 어떤 기능을 수행할 목적으로 작성하였는지 설명한다. */
abstract 리턴타입 메서드이름();

Ex)
/* 지정된 위치(pos)에서 재생을 시작하는 기능이 수행되도록 작성한다.*/
abstract void play(int pos);
```

- 꼭 필요하지만 자손마다 다르게 구현될 것으로 예상되는 경우에 사용
- 클래스 내 추상메서드가 한 개라도 존재 하면 -> 추상클래스로 선언

```
abstract class Player {
    int currentPos;          // 현재 Play되고 있는 위치를 저장하기 위한 변수

    Player() {               // 추상클래스도 생성자가 있어야 한다.
        currentPos = 0;
    }

    abstract void play(int pos);    // 추상메서드
    abstract void stop();           // 추상메서드

    void play() {
        play(currentPos);           // 추상메서드를 사용할 수 있다.
    }
    ...
}
```


주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

추상클래스 예제

```
class Marine {    // 보병
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void stimPack()         { /* 스팀팩을 사용한다.*/}
}

class Tank {      // 탱크
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void changeMode()       { /* 공격모드를 변환한다. */}
}

class Dropship {  // 수송선
    int x, y;     // 현재 위치
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stop()             { /* 현재 위치에 정지 */ }
    void load()             { /* 선택된 대상을 태운다.*/ }
    void unload()           { /* 선택된 대상을 내린다.*/ }
}
```

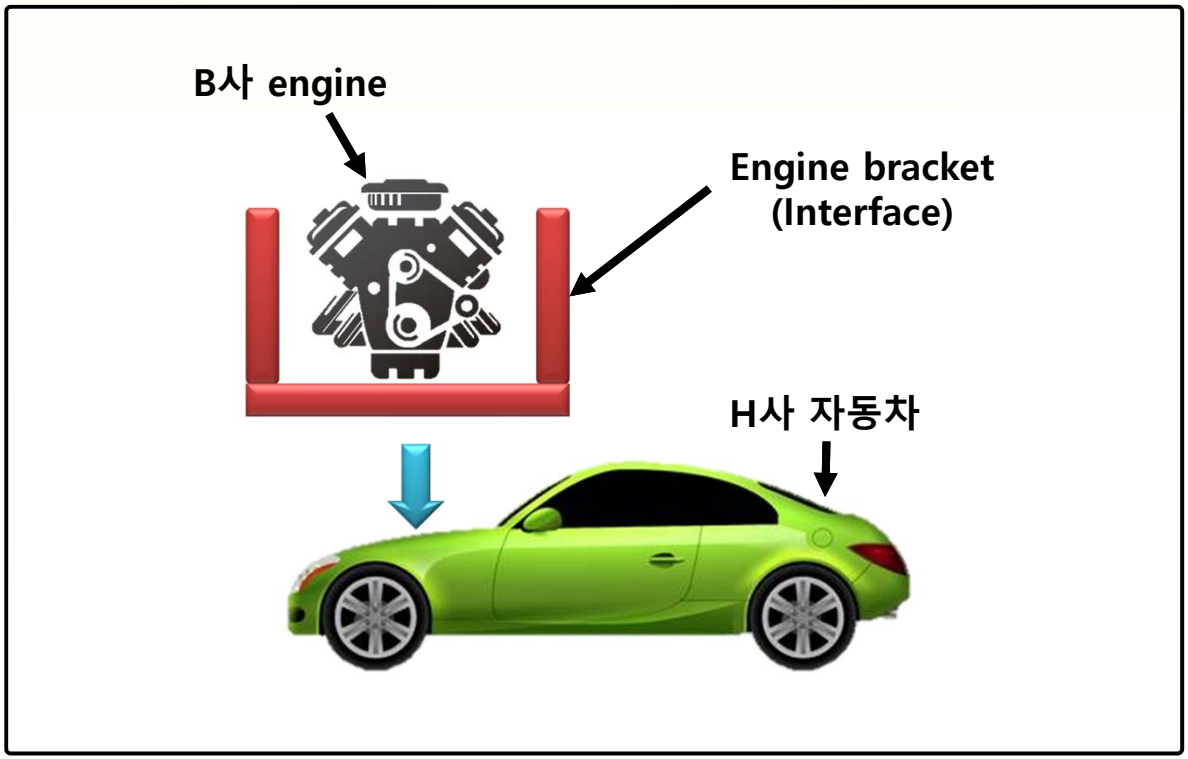
```
abstract class Unit {
    int x, y;
    abstract void move(int x, int y);
    void stop() {      /* 현재 위치에 정지 */ }
}

class Marine extends Unit {   // 보병
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void stimPack()         { /* 스팀팩을 사용한다.*/}
}

class Tank extends Unit {     // 탱크
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void changeMode()       { /* 공격모드를 변환한다. */}
}

class Dropship extends Unit { // 수송선
    void move(int x, int y) { /* 지정된 위치로 이동 */ }
    void load()             { /* 선택된 대상을 태운다.*/ }
    void unload()           { /* 선택된 대상을 내린다.*/ }
}
```

```
Unit[] group = new Unit[4];
group[0] = new Marine();
group[1] = new Tank();
group[2] = new Marine();
group[3] = new Dropship();

for(int i=0;i< group.length;i++) {
    group[i].move(100, 200);
}
```

추상메서드가 호출되는 것이 아니라 각 자손들에 실제로 구현된 move(int x, int y)가 호출된다.

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

인터페이스(Interface)


B사 engine
Engine bracket
(Interface)
H사 자동차

✓ **myEngine**
- EngineBracket 객체 참조 변수

✓ **engineOn()**
- BEngine startEngine() 호출

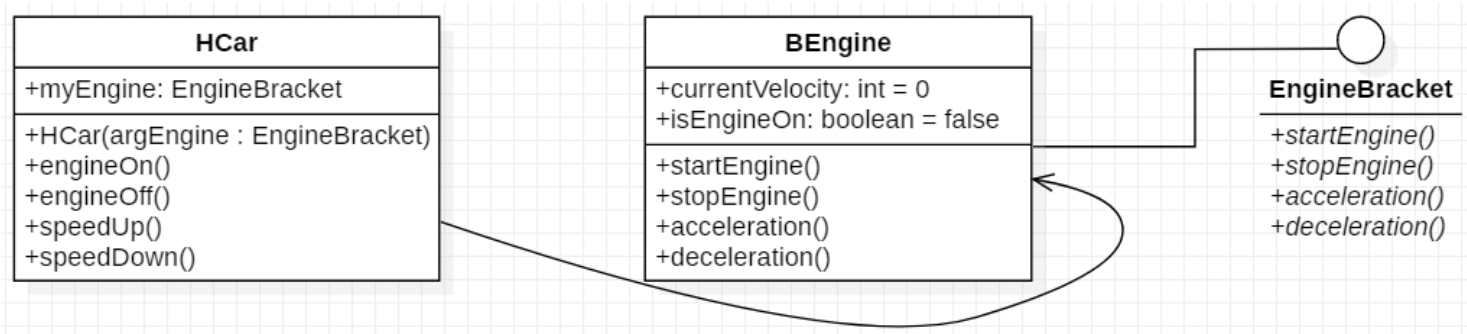
✓ **engineOff()**
- BEngine stopEngine() 호출

✓ **speedup()**
- BEngine acceleration() 호출

✓ **speedDown()**
- BEngine deceleration() 호출

✓ **currentVelocity**
- 현 엔진 속도 값 저장
- int 형, 초기값 0

✓ **IsEngineOn**
- 현 엔진의 시동 상태 값 저장
- true : engine on, false : engine off



인터페이스(Interface)란?

- 일종의 추상클래스-> 추상클래스(미완성 설계도)보다 추상화 정도가 높다

- 실제 구현된 것이 전혀 없는 기본 설계도.
  - 알맹이 없는 껍데기

- 추상메서드와 상수만을 멤버로 가질 수 있다

- 인스턴스를 생성할 수 없고, 클래스 작성에 도움을 줄 목적으로 사용된다.

- 미리 정해진 규칙에 맞게 구현하도록 표준을 제시하는 데 사용된다

- 인터페이스 간 상속이 가능 -> 다중 상속 가능

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

인터페이스(Interface) 작성

- 'class'대신 'interface'를 사용한다는 것 외에는 클래스 작성과 동일하다.

```
interface 인터페이스이름 {
        public static final 타입 상수이름 = 값;
        public abstract 메서드이름(매개변수목록);
}
```

- 하지만, 구성요소(멤버)는 추상메서드와 상수만 가능하다

```
- 모든 멤버변수는 public static final 이어야 하며, 이를 생략할 수 있다.
- 모든 메서드는 public abstract 이어야 하며, 이를 생략할 수 있다.
```

```
interface PlayingCard {
        public static final int SPADE = 4;
        final int DIAMOND = 3;      // public static final int DIAMOND = 3;
        static int HEART = 2;       // public static final int HEART = 2;
        int CLOVER = 1;             // public static final int CLOVER = 1;

        public abstract String getCardNumber();
        String getCardKind();  // public abstract String getCardKind();
}
```


주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

인터페이스(Interface)의 구현

- 인터페이스를 구현하는 것은 클래스를 상속받는 것과 같다.
- 다만, 'extends' 대신 'implements'를 사용한다.

```
class 클래스이름 implements 인터페이스이름 {
    // 인터페이스에 정의된 추상메서드를 구현해야한다.
}
```

- 인터페이스에 정의된 추상메서드를 완성해야 한다.

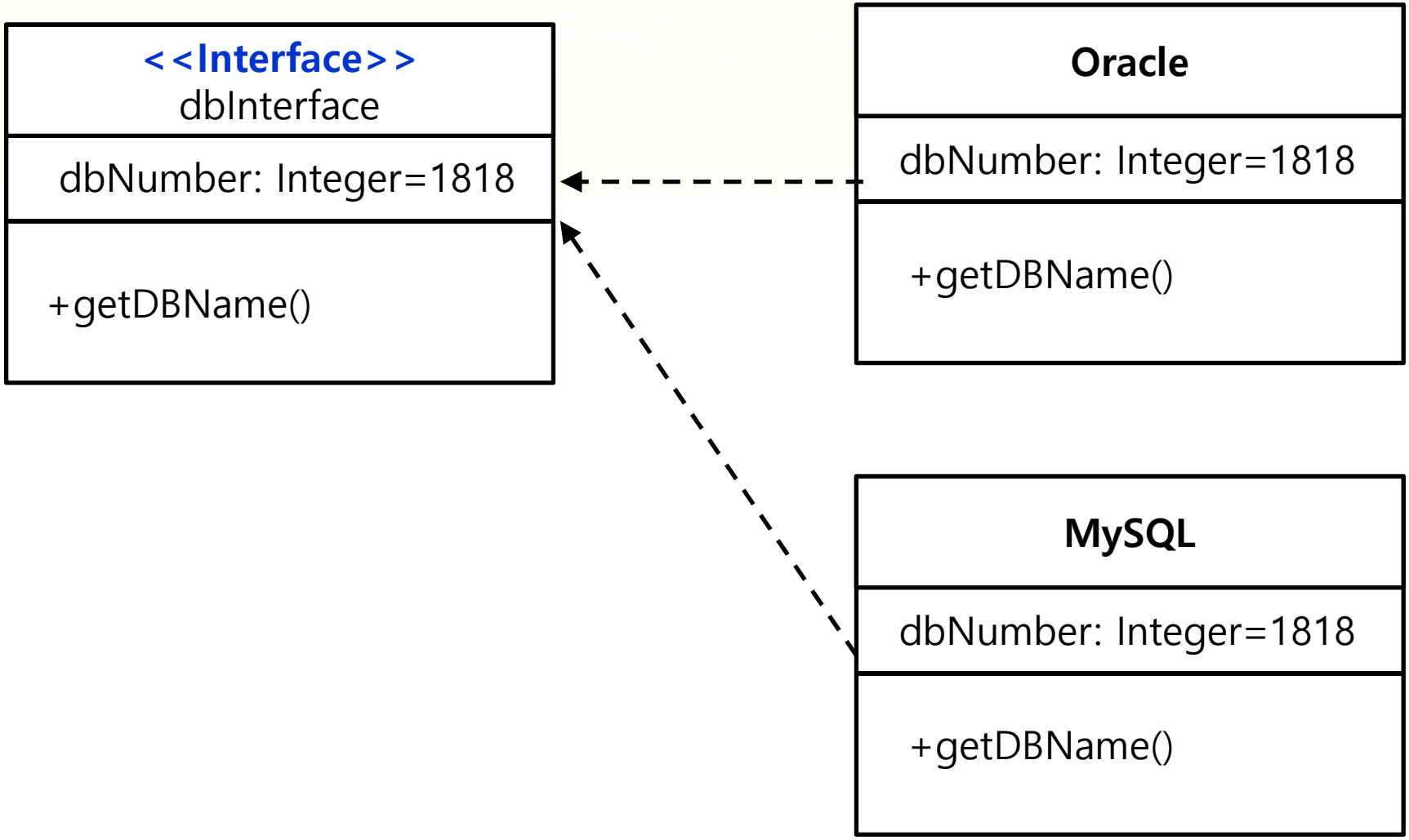
```
class Fighter implements Fightable {
    public void move() { /* 내용 생략*/ }
    public void attack() { /* 내용 생략*/ }
}

interface Fightable {
    void move(int x, int y);
    void attack(Unit u);
}
```

- 상속(extends)와 구현(implements)이 동시에 가능하다

```
class Fighter extends Unit implements Fightable {
    public void move(int x, int y) { /* 내용 생략 */}
    public void attack(Unit u) { /* 내용 생략 */}
}
```


주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

인터페이스 실습!! (정의, 구현) – 1



인터페이스 실습!! (정의, 구현) – 2

```
public class MainClass{

    public static void main(String args[]) {
        Oracle  o       = new Oracle();
        MySql   m       = new MySql();

        System.out.println(o.getDBName());
        System.out.println(m.getDBName());
    }
}
```

```
Oracle
MySql
```

인터페이스(Interface)의 상속

- 인터페이스도 클래스처럼 상속이 가능하다. -> 인터페이스 간 상속
- 클래스와 달리 다중상속 허용

```
interface Movable {
        /** 지정된 위치(x, y)로 이동하는 기능의 메서드 */
        void move(int x, int y);
}

interface Attackable {
        /** 지정된 대상(u)을 공격하는 기능의 메서드 */
        void attack(Unit u);
}

interface Fightable extends Movable, Attackable { }
```

- 인터페이스는 Object클래스와 같은 최고 조상이 없다.

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

인터페이스를 이용한 다형성

- 인터페이스 타입의 변수로 인터페이스를 구현한 클래스의 인스턴스 참조 가능

```
class Fighter extends Unit implements Fightable {
    public void move(int x, int y)  { /* 내용 생략 */ }
    public void attack(Fightable f) { /* 내용 생략 */ }
}
```

```
Fighter   f = new Fighter();
Fightable f = new Fighter();
```

- 인터페이스를 메서드의 매개변수 타입으로 지정할 수 있다.

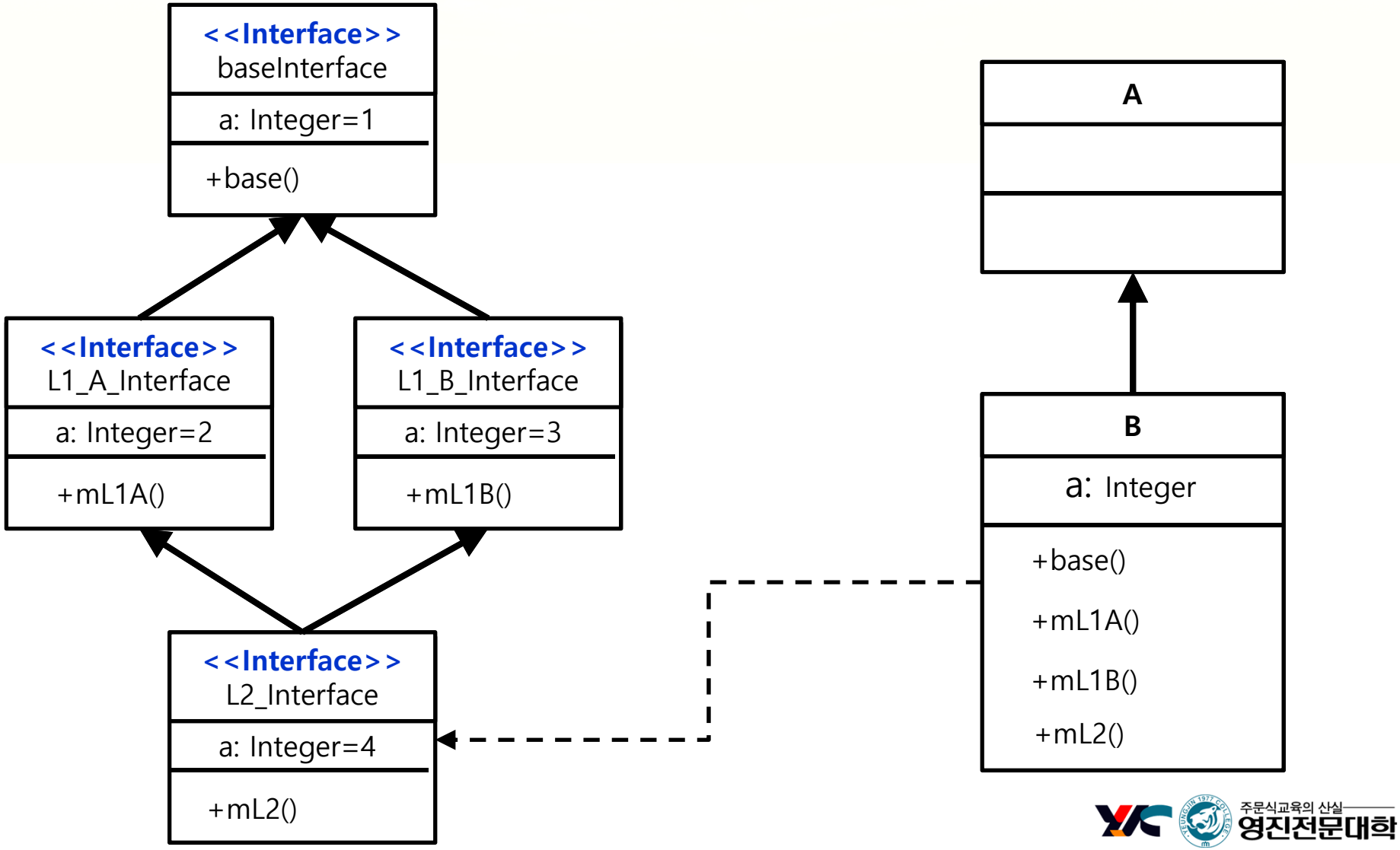
```
void attack(Fightable f) {   // Fightable인터페이스를 구현한 클래스의 인스턴스를
    //...                    // 매개변수로 받는 메서드
}
```

- 인터페이스를 메서드의 매개변수 타입으로 지정할 수 있다.

```
Fightable method() { // Fightable인터페이스를 구현한 클래스의 인스턴스를 반환
    // ...
    return new Fighter();
}
```


주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

인터페이스 실습!!(상속, 다형성) - 1



주문식교육의 산실
영진전문대학
컴퓨터정보계열 정영철

인터페이스 실습!!(상속, 다형성) – 2

```
public class MainClass{

    public static void main(String args[]) {
        baseInterface   b       = new B();
        L1_A_Interface  l1_a    = new B();
        L1_B_Interface  l1_b    = new B();
        L2_Interface    l2      = new B();
        B               myClass = new B();

        System.out.println("b.a\t\t: " + b.a);
        System.out.println("l1_a.a\t\t: " + l1_a.a);
        System.out.println("l1_b.a\t\t: " + l1_b.a);
        System.out.println("l2.a\t\t: " + l2.a);
        System.out.println("myClass.a\t: " + myClass.a);

        System.out.println("=========================

        myClass.base();
        myClass.mL1A();
        myClass.mL1B();
        myClass.mL2();

    }
}
```

인터페이스 다형성

출력결과

```
b.a             : 1
l1_a.a          : 2
l1_b.a          : 3
l2.a            : 4
myClass.a       : 4
=========================
base()  method is invoked
mL1A()  method is invoked
mL1B()  method is invoked
mL2()   method is invoked
```

인터페이스 실습!!(상속, 다형성) – 3

```
interface baseInterface {
    int  a = 1;
    void base();
}

interface L1_A_Interface extends baseInterface{
    int a = 2;
    void mL1A();
}

interface L1_B_Interface extends baseInterface{
    int a = 3;
    void mL1B();
}

interface L2_Interface extends L1_A_Interface, L1_B_Interface{
    int a = 4;
    void mL2();
}
```

인터페이스 실습!!(상속, 다형성) - 4

```
class A {
}

class B  extends A implements L2_Interface{
    public void base() { System.out.println("base()\tmethod is invoked"); }
    public void mL1A() { System.out.println("mL1A()\tmethod is invoked"); }
    public void mL1B() { System.out.println("mL1B()\tmethod is invoked"); }
    public void mL2()  { System.out.println("mL2()\tmethod is invoked");  }
}
```

인터페이스의 장점?

- 개발시간을 단축시킬 수 있다.
- 표준화가 가능하다
- 서로 관계없는 클래스들에게 관계를 맺어 줄 수 있다.
- 독립적인 프로그래밍이 가능하다.