10장. 객체지향 - 다형성

1. 다형성이란?

'다형성'은 여러 개의 서로 다른 형식과 모양을 가진다는 의미로, 다형성을 이용하면 객체의 유형에 따라 서로 다른 작업을 수행하도록 할 수 있다.
객체는 한가지 타입에 하나의 형태만을 생성하는 고정적인 형태보다, 한가지 타입으로 여러 형태를 생성할 수 있는 유동적인 형태가 프로그램을 더욱 편하게 작성하게 해준다.
레퍼런스가 수퍼클래스이고, 생성되는 객체가 서브클래스가 할당되면, 할당되는 서브클래스의 객체 타입에 따라 다양한 형태의 객체를 생성할 수 있다. 자바에서는 다형성을 추상클래스와 인터페이스를 이용하여 구현한다.
개체는 각각 자신의 책임으로 작동한다. 즉, 인수 등으로 동작을 특정하지 않아도 자신이해야 할 일을 객체가 알고 있는 것이다. 이것을 다형성이라고 한다.
개체를 슈퍼 클래스 형이나 인터페이스 형에 대입하여 메소드 호출을 하면 각각 생성 된 클래스 형에 따라 적절하게 작동한다. 즉, 같은 클래스, 인터페이스 형 개체에 대해 같은 메소드를 호출해도, 그 객체가 생성 된 원래의 클래스에 따라 다른 동작을 하는 것이다.
예를 들어, Drawable 인터페이스에 draw () 메소드를 정의하고 이를 Rectangle, Circle, Triangle에서 구현한다. 이때 Rectangle 객체와 Circle 객체는 Drawable 형 변수에 할당 할 수 있다

```
Drawable [] shapes = {new Rectangle (), new Circle (), new Triangle ()};
```

이러한 구현 클래스에서 draw ()가 별도로 구현되어 있으며, 그 동작은 모두 다르다. Drawable 형 변수 shapes에 draw ()를 호출하면 할당 된 객체의 원래 클래스에서의 구현에 따라 다르게 작동할 것이다.

```
shapes [0] .draw (); // 사각형을 그린다
shapes [1] .draw (); // 동그라미를 그린다
shapes [2] .draw (); // 삼각형을 그린다
```

이것이 다형성이다.

2. 추상 클래스와 추상 메소드

응용 프로그램의 설계를 할 때 여러 클래스로 구성되도록 한다 객체 지향에서는 여러 객체가 메시지를 교환하여 처리가 진행되므로 하나 하나의 개체의 기능은 모드 계획에 따라 선언하는 것이 확장 성이 풍부하고, 재사용하기 쉬워진다. 또한 모든 클래스를 모두 구현하는 것이 아니라 적절하게 계승하고 중복 기능을 두 번 구현 하지 않도록 한다. 이 때 핵심 공통 기능 만 구현 한 클

래스를 준비하고 실제로 상속 클래스를 인스턴스화하고 개체를 이용하도록 하는 것이 다형성의 설계이며 그 중 하나가 추상 메소드를 가진 추상클래스이다.
추상 메소드를 하나라도 가진 클래스는 추상 클래스이며 인스턴스화 할 수 없다. 추상 클래스 / 추상 메소드 모두 한정자에 abstract를 지정해야 하며 추상 메소드는 상속 클래스에서 구현해야 하지만, 구현 시에는 abstract 한정자 이외는 모두 동일하게 지정해야 한다.
추상 클래스는 인스턴스화 할 수 없지만 추상 클래스 형의 변수를 선언 할 수 있다. 추상 클래스를 사용하는 경우 여러 하위 클래스로 상속되는 것을 명시하고 있으며, 그 형태의 객체 (참조 ID)를 모두 대입 할 수 있으므로 편리하다.
예를 들면 어떤 기업에서 다음달부터 김치 버거를 판매한다라고 하자. 본사에서 김치 버거의 레시피, 버거의 명칭, 구성 및 가격 등을 선언해 놓으면 각 프랜차이즈는 본사의 김치 버거에 대한 선언된 내용을 가져다가 레시피에 따라 김치 버거를 직접 만들고 명칭을 광고하고 구성 및 가격에 따라 판매량을 계산하게 된다. 이때 본사에서 선언된 내용들이 아직 가공되어 만들어 지지 않은 선언한 상태의 추상 메소드들이 되고 이러한 추상 메소드를 가진 본사는 추상 클래스가 된다. 본사는 각 프랜차이즈 지점에서 재정의된 메소드들을 통해 김치 버거의 판매량를 지점별로 관리할 수 있는 장점과 또한 각 다른 지점에 있는 모든 프랜차이즈 들은 같은 김치 버거를 같은 레시피로 같은 가격에 판매할 수 있는 네임 벨류를 가진다. 물론 각 지점들의 판매량은 모두 같지 않은 다양한 상태를 가진다. 메소드는 후손이 모두 통일하되 각 리턴되는 내용을 다르게 받을 수 있는 장점이 다형성이다.
또 다른 예를 들면 본사에서 각 지사의 모든 대리들을 모아 회의를 한다면 명칭은 모두 대리지만 하는 일은 모두 다르다. 다형성의 단편적인 예이다.
추상클래스는 상속 시 다형성을 유도하도록 선언되는 클래스로 동적 바인딩을 이용한 구문이다. 다음과 같은 특징을 가진다.

abstract 라는 키워드를 클래스 앞, 메소드 앞에 각각 선언하면 추상클래스 추상 메소드가 된다.

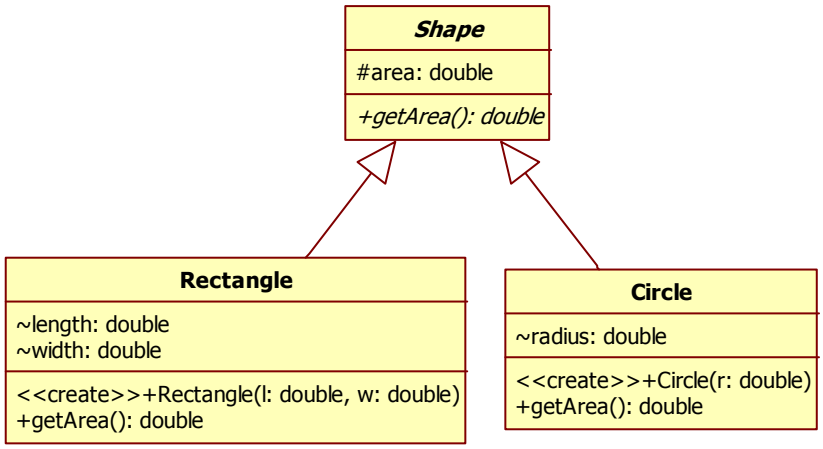
```
public     abstract class AA{    ------→추상클래스
public abstract void prn();     -----→추상 메소드
}
```

추상 메소드는 선언만 있고 바디 부분{} 즉 명령 부분을 명시하지 않는다.
추상 클래스라고 선언을 하게 되면 추상클래스를 상속받은 하위 클래스는 추상 클래스의 추상 메소드를 반드시 정의 하여야 한다.
추상 메소드를 재정의하지 않는 후손 클래스는 추상 클래스가 된다. 추상 클래스는 후손의 재정의를 위한 클래스이므로 객체 생성은 불가능 하며 주소 참조 변수로는 사용 할 수 있다.

```
new AA()     ->불가능
AA   a1=new BB(); // 후손의 BB클래스의  생성을 통해 주소 참조 가능
```

추상 클래스도 클래스이기 때문에 생성자 및 멤버들을 클래스와 동일하게 가진다.
자바에서는 추상 메소드를 가지지 않아도 추상클래스를 선언해서 사용할 수 있다.

다음 프로그램을 살펴 보자. 수퍼 클래스인 Shape가 레퍼런스가 되고 서브 클래스인 Rectangle, Circle클래스가 메모리 생성되어 할당되면 재정의된 getArea()메소드를 통해 각 도형의 넓이를 리턴 받아 사용할 수 있다. Shape와 getArea()가 이탤릭으로 놓여 있는것은 다이어그램에서 추상 클래스를 나타낸다.




```
abstract class Shape {
protected double area;
public abstract double getArea();
}

class Rectangle extends Shape {
double length;
double width;
public Rectangle(double l, double w) {
length = l;
width = w;
}
public double getArea() {
area = length * width;
return (area);
}
}
class Circle extends Shape {
double radius;
public Circle(double r) {
radius = r;
```

```
}
public double getArea() {
area = 3.14 * (radius * radius);
return (area);
}
}
public class TestPolymorphism {
public static void main(String[] args) {
System.out.println("The area is: ");
Shape[] my_shape = new Shape[] { new Circle(5), new Rectangle(4, 5) };
for(Shape s : my_shape){
if(s instanceof Circle){
System.out.println("Circle area :"+s.getArea());
}else{
System.out.println("Rectangle area :"+s.getArea());
}
}

}

}
```

[실행결과]

```
The area is:
Circle area :78.5
Rectangle area :20.0
```

3. 인터페이스(interface)

Java와 같은 객체 지향 언어에서는 데이터 나 메소드를 숨기는 캡슐화 (encapsulate)가 중요하다. 데이터는 모든 private 자격을 외부에서 은폐하고 public 자격이 된 메소드를 통해 접근한다. 외부에 공개 할 필요가 없는 처리의 구현도 private 자격 메소드 내에 기술하는 것으로, 클래스 외부에서 은폐한다. 그런다. 외부에 최저 한도의 메소드만을 public 으로 공개하여 이 공개 된 메소드는 해당 클래스를 인스턴스화 한 개체에 다른 개체가 액세스 할 수 있는, 즉 인터페이스가 되는 것이다.

인터페이스는 음식점의 메뉴판과 기본 메뉴와 같은 것이다. 메뉴판을 보고 고객이 원하는 음식을 요청하게 되는데 메뉴판 자체가 음식을 주지는 않는다. 실제 음식은 주방이라는 곳에서 나오므로 메뉴판은 고객이 호출할 수 있는 서비스의 목록이라 할 수 있다. 또한 기본적인 메뉴가 특성화 되었다. 예를 들면 라면집에 가면 기본라면이 있고 매운 라면, 달콤한 라면, 치즈 라면 등이 있듯

기본 메뉴가 있다. 이렇듯 인터페이스는 선언할 때 멤버를 기본 메뉴는 default 메소드와 메뉴는 추상 메소드 가격은 상수라고 비교하여 생각한다면 선언 시 어떤 멤버가 올지 유추할 수 있다.
다음은 인터페이스의 선언형식이다 interface 모든 멤버의 접근 지정자는 public 이다.

```
public interface <인터페이스 이름> {
상수;
추상메소드;
defalt 메소드;
}
```

다음은 선언 예이다.

```
interface Interface1 {
// 필드
int INT_VAL1 = 10;
// 추상 메소드
String method (int a, int b);
default String  nickname{
return “야옹이”;
}
}
```

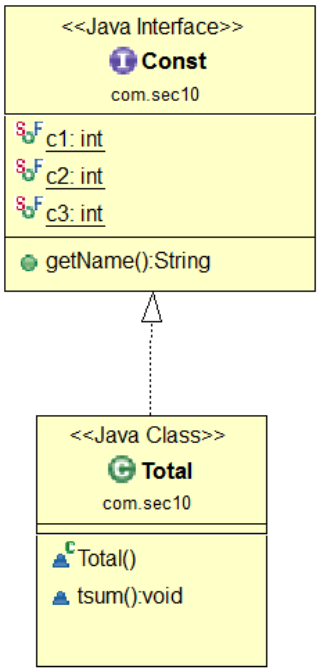
자바는 다중상속을 지원하지 않는다.
자바에서 간접적인 다중상속을 위해 인터페이스를 제공하는데 인터페이스는 기본적으로 멤버변수와 추상메소드의 집합이다. 인터페이스는 스펙(spec)만을 기술해 놓은 것으로 인터페이스를 사용하려면 클래스에서 인터페이스를 구현하여야 한다.
즉, 클래스가 인터페이스를 구현하도록 선언하고, 인터페이스 내에 선언된 각 메소드의 코드를 클래스 정의의 일부로 작성해 넣어야 한다.
인터페이스가 가지고 있는 메소드가 모두 추상 메소드이므로 인터페이스는 구현하는 클래스에서 해당 메소드를 반드시 모두 오버라이딩 해야 한다는 것이다.
인터페이스에 들어있는 메소드는 항상 public abstract이다. 즉, public abstract를 지정하지 않아도 자동으로 이들을 갖는다.
인터페이스의 멤버변수는 항상 public static final이다. 이것도 기본값으로 정해져 있으므로 굳이 선언하지 않아도 된다. 또한 인터페이스는 public abstract의 속성을 갖는다.
인터페이스는 interface 라는 키워드로 정의한다. 클래스의 경우 하나의 클래스 만 상속 (extends) 수 있지만, 인터페이스의 경우 여러 인터페이스를 구현 (implements) 할 수 있다.

```
class InterfaceImpl implements Interface1, interface2, interface3 {
...
}
```

인터페이스와 인터페이스끼리의 상속 - **extends**

클래스(후손)와 인터페이스 (선조) 상속 - **implements**

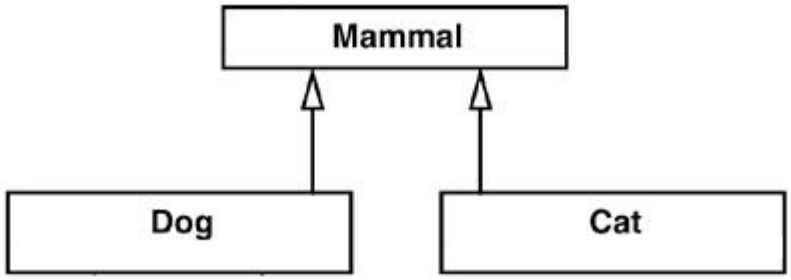
다음 예제는 두 클래스와 하나의 인터페이스가 정의되어있다. 인터페이스는 변수 정의만 행해, 구현 클래스에서 그 총합과 평균을 출력한다.



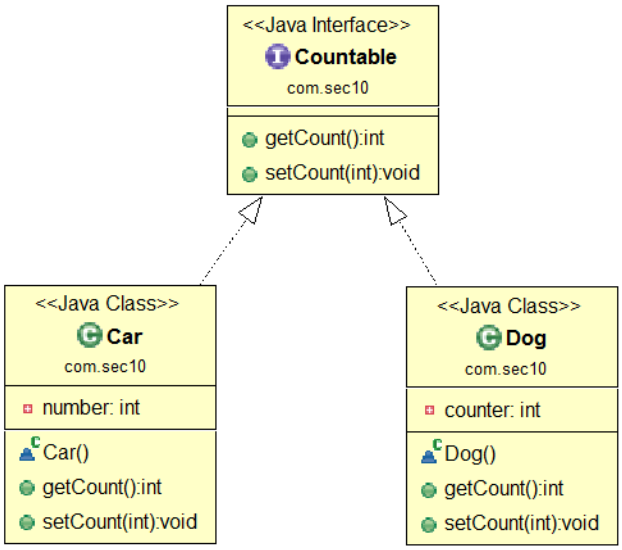

```
interface Const {
int c1 = 85, c2 = 94, c3 = 72;
default String getName(){
return "멍멍이";
}
}
class Total implements Const {
void tsum () {
int tot;
tot = c1 + c2 + c3;     -->implements구현시 멤버를 모두 호출 가능
System.out.println (getName()+"의 Total :"+ tot);
default메소드는 제정의 없이 호출
tot /= 3;
System.out.println (getName()+ "의 Avg  :"+ tot);
}
}
public class TestInterface {
public static void main(String[] args) {
Total obj = new Total ();
obj.tsum ();
}
}
```

[실행결과]
멍멍이의 Total :251
멍멍이의 Avg :83

>> is~a
일반 클래스의 경우는 인스턴스화에 의해 만들어진 객체를 참조하는 ID를 할당하는 변수를 클래스 형 변수로 선언한다.
인터페이스 형의 변수도 정의 할 수 있다. 인터페이스 형으로 선언 된 참조 형 변수는 그 인터페이스를 구현하는 클래스에서 만들어진 모든 개체 참조를 할당 할 수 있다.
클래스 형 변수의 경우 해당 클래스에서 생성 된 객체뿐만 아니라, 서브 클래스에서 생성 된 객체를 할당 할 수 있다. 인터페이스 형 객체의 경우, 구현 클래스에서 생성 된 객체를 할당 할 수 있다. 이러한 의미에서, 슈퍼 클래스의 상속과 인터페이스 구현은 모두 "is-a"관계에 있다라고 한다.
예를 들어, 포유류 클래스 Mammal 을 계승하고 고양이 클래스 Cat 을 만드는 경우 "고양이는 포유류이다."(A Cat is a Mammal)라는 관계에 있다 불리는 객체가 생성 된 클래스를 식별하는 인터페이스 Nameable 을 구현하고 고양이 클래스 Cat 을 만드는 경우 "고양이는 명명 가능하다."(A Cat is Nameable)라는 관계에 있는 것이다. 물론 Doge도 "강아지는 명명가능하다"의 관계에 잇다. "is-a"관계는 '~은 ~이다', '~은 ~의 일종이다 "라는 관계를 말한다. 이것은 컴파일러가 "~와 ~는 대입 호환하다"고 인식하는 것을 의미한다. 즉, 하위 클래스 형식 개체는 슈퍼 클래스 형 변수에 할당 할 수 있다. 구현 클래스 형 객체는 인터페이스 형 변수에 할당 할 수 있다.



다음은 인터페이스 Countable을 구현하는 두 가지 클래스 Car와 Dog를 인스턴스화하고 Countable 형 변수에 대입 인터페이스 메소드의 사용을 구현하고 있다 is~a 의 관계로 구현되는 것을 확인할 수 있다.




```
interface Countable {
int getCount ();
void setCount (int aCounter);
}
class Dog implements Countable {
private int counter;
public int getCount () {
return counter;
}
public void setCount (int aCounter) {
counter = aCounter;
}
}
class Car implements Countable {
private int number;
public int getCount () {
return number;
}
public void setCount (int aCounter) {
number = aCounter;
}
}
public class TestInterface01 {
public static void main(String[] args) {
Dog dog = new Dog ();
Car car = new Car ();
```

```
Countable [] objs = {dog, car}; --------→ // is-a 관계


objs [0] .setCount (10); ----------------→인터페이스 메소드의 사용
objs [1] .setCount (2);

for (int i = 0; i <objs.length; i++) {
System.out.println (objs [i] .getCount ()); ------→인터페이스 메소드의 사용

}
}
}
```

[실행결과]

```
10
2
```

4. 스택틱(static)과 파이널 (final)

스태틱(static)이란 정적이란 뜻을 가지고 있으며 멤버변수 또는 메소드 선언문에 사용되어 해당 멤버 변수 또는 메소드가 클래스 로딩 시 자동 생성됨을 나타내는 키워드이다. 스태틱 멤버 변수 또는 메소드는 오브젝트 생성 전에 이미 메모리에 생성되므로 new 키워드를 이용해서 해당 클래스의 오브젝트를 메모리에 생성하더라도, 매번 생성되는 것이 아니라 클래스 로딩시 생성된 멤버 변수 또는 메소드를 계속 사용하게 된다. 스태틱 변수와 스태틱 메소드는 각각 클래스 변수, 클래스 메소드라고 부르기도 한다.
파이널(final)이란 final은 클래스, 메소드, 변수(local 변수, 멤버변수)에 쓸 수 있는 키워드이다.
final로 선언된 클래스는 더 이상 상속될 수 없다.
<예> java.lang.String class는 final class로 상속되지 않는다.
final로 선언된 Method는 override될 수 없지만 overload는 가능하다
파이널 변수의 초기화는 다음 두 가지로 이루어 진다.

① 변수 선언과 동시에 딱 한번 초기화
public final int finalVar = 1;
② static 블록에서 딱 한번 초기화

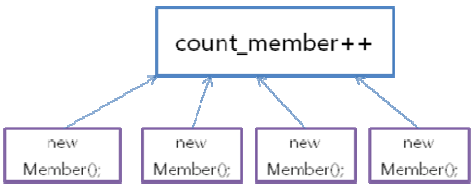
```
public final int finalVar; // 선언시 초기화 하지 않고,
static { //static 블록에서 초기화
finalVar = 5;
}
```

static 키워드가 변수, 메소드 앞에 선언되면 스태틱 변수와 메소드가 되며 객체 생성 없이 클래스명. 멤버로 호출 해서 사용한다 특징은 다음과 같다.
자바에서 static 키워드는 한 클래스의 모든 객체들이 변수를 공유할 때 사용된다.
static으로 선언된 변수 혹은 static 메소드는 클래스의 객체가 존재하지 않더라도 사용할 수 있다.
스태틱 메소드내에서 사용될 수 있는 외부 변수는 스태틱으로 선언한 변수이어야 한다.
스태틱 메소드내에서 this나 super를 사용할 수 없으며 static 키워드는 c++처럼 지역 변수에 대해 선언할 수 없다.
스택틱의 용도는 다음과 같은 경우에 사용된다

.

공통 메소드 를 정의하는 경우
· 일반 상수를 정의하는 경우
· 여러 클래스간에 공통 값을 공유하려는 경우

다음은 스태틱 변수를 사용한 프로그램이다.



static 변수인 count_member++
변수는 객체가 클래스오브젝트에 단 한번만
존재 하는 변수이다.
new Member()객체가 생성될 때 마다
변수 값이 1씩 증가 되어 결과가 4가 출력되는
것을 볼 수있다.

```
class Member {

static int count_member = 0;     //------------->초기화
String name;

public Member(String name) {
this.name = name;
count_member++;   //--------->객체를 생성할 때 마다 증가.
}
}
```

```
public class TestStatic {

public static void main(String[] args) {
for (int i = 0; i < 4; i++) {
new Member(i + "[번 째 멤버 ]");
}
System.out.println("오늘 입장한 멤버 수 : " + Member.count_member);
}

}
```

[실행결과]
오늘 입장한 멤버 수 : 4

스태틱의 초기화는 사용되는 클래스의 특정 부분을 초기화 해 주기 위해서 static block을 사용한다. 단, 이러한 static block은 메소드 내에서 사용될 수 없다.
class가 사용되기 위해 로드 되어질 때, 한번 실행 되어지며 사용되지 않는 클래스는 로드되지 않는다.
static초기화 블럭은 클래스가 초기화될 때 수행되고, main() 메소드보다 먼저 수행된다.

```
public class StaticInitTest {

    static int i = 10;
    static {
        System.out.println("Static i=" + i++);
    }

    public static void main(String[] args) {
        System.out.println("Main  " + StaticInitTest.i);
    }
}
```

```
Static i=10
Main  11
```