< 다형성 >

1. 다형성

- **상속 관계에 있는 클래스 일 때 부모타입이 모든 후손 타입의 주소를 받을 수 있음**
- 상속을 이용한 기술로써 여러 종류의 클래스 타입을 처리하는 기술
- 매개변수 하나가 여러 종류의 클래스 타입을 처리할 수 있는 기능

➔ 후손 객체 안에 부모 멤버가 포함되어 있기 때문에 가능함 => 클래스 형 변환

ex)

```
// A가 부모클래스이고, B와 C가 자식클래스일 때
A aobj = new B();
aobj = new C();
```

➔ 동일한 기능을 처리하는 메소드를 여러 개 작성해야 할 경우(하지만 처리할 자료형이 다를 경우) : **오버로딩** 사용

→ 오버로딩 메소드의 개수를 줄일 수 있는 방법으로 다형성 이용

→ 처리해야 할 여러 종류의 자료가 서로 상속관계에 있다면, 메소드 작성 시 부모 클래스형 레퍼런스를 매개변수로 작성하면, 여러 후손 클래스 타입을 하나의 메소드가 다 처리할 수 있음

ex)

```
// 다형성을 적용 안 하면 아래 두 개의 오버로딩 메소드로 구현
public void method(B b);
public void method(C c);
// 다형성을 적용 하면 아래와 같이 한 개의 메소드로 처리할 수 있음
public void method(A a);
```

➔ **따라서 오버로딩(중복 작성)을 줄이면서, 원하는 클래스 타입들을 처리하고자 할 경우 다형성을 이용함**

< 클래스 형변환 >

```
public class Parent { }                    // 부모 클래스

public class Child1 extends Parent { }     // 자식 클래스

public class Child2 extends Parent { }     // 자식 클래스
```

// -------- **Child1, Child2 클래스형 → Parent 클래스형으로 바뀜** --------//

*** 자동 형변환 적용됨 : 하위 클래스형이 상위 클래스형에 대입될 경우**

후손 타입이 부모타입으로 바뀌면서 대입됨 → **UpCasting**

자동 형변환 → **명시할 필요 없음**

```
Parent p;

Child1 c1 = new Child1();

Child2 c2 = new Child2();


p = c1;

p = c2;
```

//----------//

// ---------- **Parent 클래스형 → Child1 클래스형으로 바뀜** ----------//

*** 강제 형변환 적용함 : 상위 클래스형을 하위 클래스형에 대입하고자 하는 경우**

강제적, 명시적으로 형변환을 해야됨 → DownCasting

((후손클래스형)부모레퍼런스)

```
Parent p = new Child1();

Child1 c1 = (Child1)p;
```

//----------//

!!!!! 클래스 간의 형변환은 반드시 상속관계에 있는 클래스끼리만 가능함 !!!!!

< instanceof 연산자 >

- 부모 레퍼런스로 다양한 객체들의 주소 정보를 받았을 경우
- **이 레퍼런스(부모)가 어떤 자식 주소 값을 참조하고 있는지 비교할 때 사용**
- 주로 DownCasting을 해야 할 때 사용

ex)

```
if(부모레퍼런스 instanceof 자식클래스형){
        // DownCasting을 통해 자식 객체에 접근하여 자식 메소드 실행
        ((자식클래스형)부모레퍼런스).자식멤버~~;
}
```

< 동적바인딩 >

- 프로그램이 실행되기 전에 컴파일이 되면서 모든 메소드는 정적 바인딩이 되는데 프로그램이 실행되면서 **해당 레퍼런스가 참조하고있는 객체타입을 기준으로 바인딩 됨**
- 즉 정적바인딩을 통해서는 해당 변수 타입의 메소드를 실행하는데 **만약 후손클래스에 실행하고자 하는 메소드가 오버라이딩 되어있을 경우 후손클래스의 메소드가 실행**
  → 다운캐스팅을 명시적으로 할 필요가 없다.