다형성 [Java]

소유자	종수 김
태그	

[다형성]

[시작]

객체지향 프로그래밍의 대표적 특징

- 캡슐화
- 상속
- **다형성 - 객체지향의 꽃**

[다형성 - Polymorphism]

다양한 형태, 여러 형태, 프로그래밍에서 다형성은 여러 타입의 객체로 취급될 수 있는 능력

[다형적 참조] - 부모 타입의 변수가 자식 인스턴스 참조

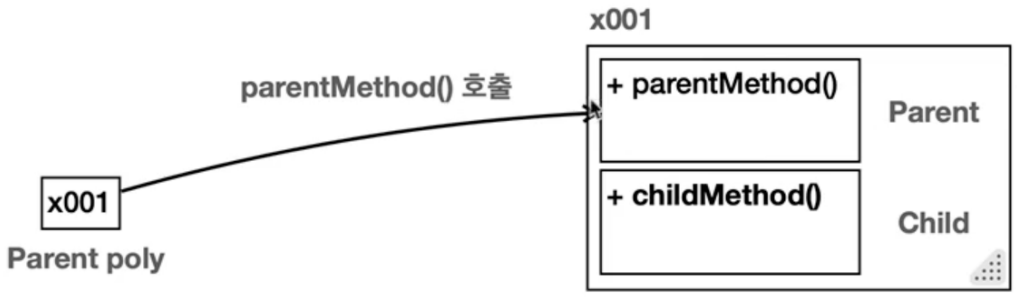
부모 타입은 자식 타입을 담을 수 있다.

자식은 부모가 가지고 있는 모든걸 가지고 있고, 추가적으로 본인 것을 가지고 있기에 부모가 자식을 담을 수 있고, 자식이 부모를 담을 수 없는 것.

- 부모 타입의 변수가 부모 타입의 인스턴스를 생성
  - 메모리상에 부모 인스턴스만 생성
- 자식 타입의 변수가 자식 타입의 인스턴스를 생성
  - 메모리상 같은 참조 값에 부모, 자식 인스턴스 생성 - 참조 값은 자식 인스턴스를 바라보고 있음.
- 부모 타입의 변수가 자식 타입의 인스턴스를 생성
  - 메모리상 같은 참조 값에 부모, 자식 인스턴스 생성 - 참조 값은 부모 인스턴스를 바라보고 있음.
  - 하지만, 부모 타입의 변수로 생성 되었기에 자식 인스턴스의 값은 사용할 수 없음.
  - 인스턴스에 오버라이딩 되어있어도 부모 타입의 함수 사용.

다형적 참조: 부모 타입의 변수가 자식 인스턴스 참조

Parent → Child: poly.parentMethod()



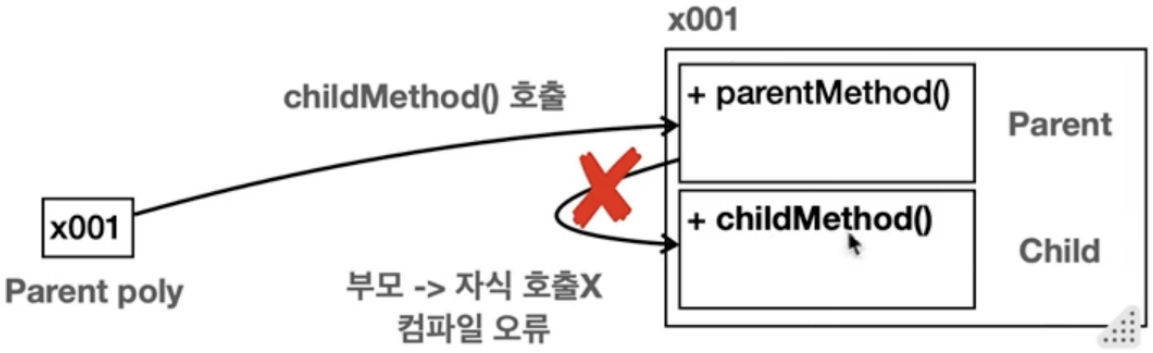
- 부모 타입의 변수가 자식 인스턴스를 참조한다.

[다형적 참조와 인스턴스 실행]

poly.parentMethod()를 호출하면, 참조값을 사용해서 인스턴스를 찾고, 다음으로 인스턴스 안에서 실행할 타입을 찾는데 Parent 타입에 parentMethod가 존재하므로 메서드 호출

[다형적 참조의 한계]

Parent → Child:poly.childMethod()



poly는 Parent 타입으로 **Parent 클래스부터 시작해서 필요한 기능을 찾는데, 상속 관계는 부모 방향으로 올라갈 수는 있으나, 자식 방향으로 내려갈 수는 없다.** 때문에 상위에 부모가 없고, 메서드가 없으므로 컴파일 오류가 발생하는 것.

- 이런 경우 '캐스팅'이 필요하다.

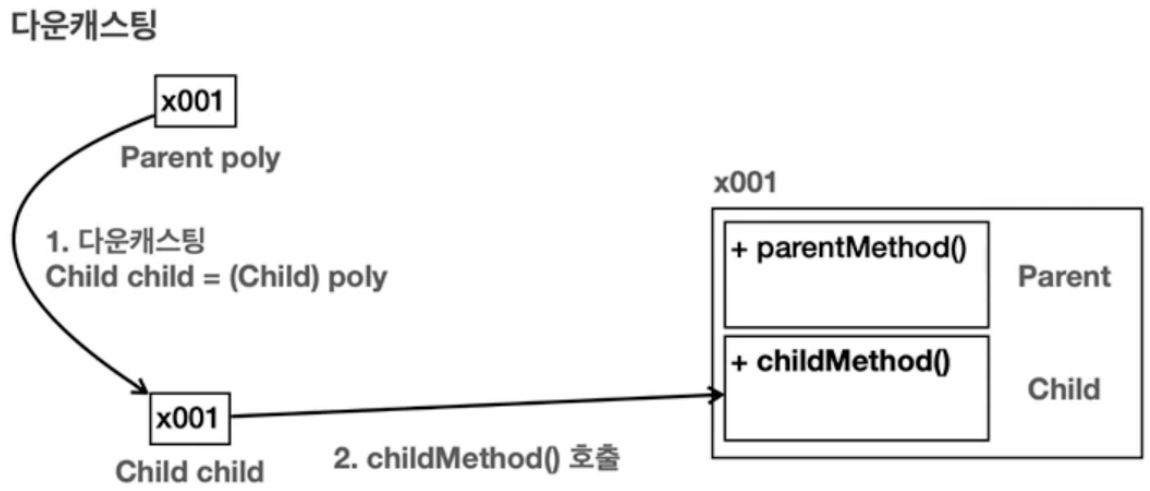
> 다형적 참조의 핵심은 부모는 자식을 품을 수 있다는 것.

[Q. 다형적 참조가 왜 필요할까?]

[다형성과 캐스팅]

Parent poly = new Child() 는 부모 타입의 변수로 자식 인스턴스의 메서드를 사용할 수 없다.

[다운캐스팅]



원래라면, 부모 타입의 변수로 선언 되었다면 자식 타입의 인스턴스를 사용할 수 없으므로 컴파일 에러를 발생시킴.
다운캐스팅이란,
강제로 부모 타입의 변수를 자식 타입의 변수로 캐스팅하는 것.

'메모리에 동시에 존재하는 Parent, Child중 참조를 잠시 Child로 변경한다'

- 캐스팅을 했다고 해서, poly자체 타입이 변하진 않음.
  단순 해당 참조값을 Child타입으로 꺼냈던것이고 poly 변수와는 상관이 없음.

* 업 캐스팅 : 부모 타입으로 변경
* 다운 캐스팅 : 자식 타입으로 변경

[캐스팅의 종류]

일시적 다운 캐스팅 : 캐스팅의 과정이 번거로움으로 해당 메서드를 호출하는 순간만 다운캐스팅

```
((Child) poly).childMethod();
```

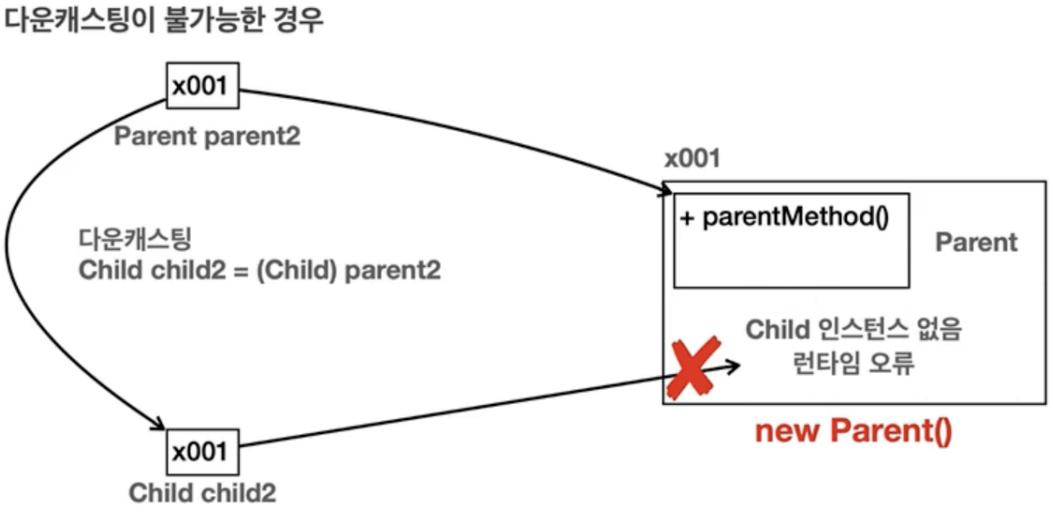
업 캐스팅 : 다운캐스팅과 반대로, 현재 타입을 부모타입으로 변경하는 것.

```
Child child = new Child();
Parent parent = (Parent) child; // 생략 가능
```

Q. 업 캐스팅은 생략이 가능한데, 다운 캐스팅은 생략하지 못하는 이유는 ?

A. 다운 캐스팅은 잘못하면 심각한 런타임 오류를 뱉을 수 있기 때문.

```
package poly.basic;

//다운 캐스팅을 자동으로 하지 않는 이유
public class CastingMain4 {
    public static void main(String[] args) {
        Parent parent = new Child();
        Child child = (Child) parent;
        child.childMethod(); // 문제 없음.

        Parent parent1 = new Parent();
        Child child1 = (Child) parent1;
        child1.childMethod();
    }
}
* parent1은 참조값에 Child 인스턴스가 존재하지 않음.
Parent객체는 childMethod()를 가지고 있지 않으므로 예외가 터지는 것.
Runtime시점까지, 힙 메모리에 인스턴스가 존재하는지 안하는지 알 수 없음.
ClassCastException
```



- 업 캐스팅은 안전하고, 다운 캐스팅은 위험한 이유

업캐스팅은 이런 문제가 절대로 발생하지 않는다. 객체를 생성하면 해당 타입의 상위 부모타입은 모두 함께 생성되기 때문에. Runtime시점에 힙 메모리에 인스턴스가 무조건 생성됨이 보장.

다운캐스팅은 부모 타입으로 선언 시, 자식 인스턴스가 힙 메모리에 같이 존재하지 않음. 때문에 다운캐스팅을 시도했을 때, Runtime시점에 Heap영역에 자식 인스턴스가 없는 경우가 존재할 수 있기 때문.

- Heap 영역에 인스턴스는 Runtime시점밖에 알 수가 없음. 때문에 다운캐스팅의 Excpetion 발생은 런타임 시점까지 알 수 없고, 이를 개발자가 확실히 처리하고 가야할 문제이기 때문에, 다운캐스팅은 강제하도록 하는 것.
- 컴파일 오류 : 변수명 오타, 잘못된 클래스 이름등 컴파일 되기 전 알 수 있는 오류.
  런타임 오류 : 프로그램이 실행되고 있는 시점에 발생하는 오류.

[instanceof]

다형성에서 참조형 변수는 다양한 자식을 대상으로 참조할 수 있으므로, 어떤 인스턴스를 참조하고 있는지 확인.
현재 참조하고 있는 인스턴스가 어떤 인스턴스인지 판단.

```
if (parent instanceof Child){
  //child class method
}
```

- 오른쪽 대상의 자식 타입을 왼쪽에서 참조하는 경우에도 true를 반환.
  부모는 자식을 담을 수 있는 경우에도 True
- instanceof를 통해 인스턴스가 정확히 다운캐스팅이 되는 상태인지 확인하고, 캐스팅을 하는 것이 좋음.

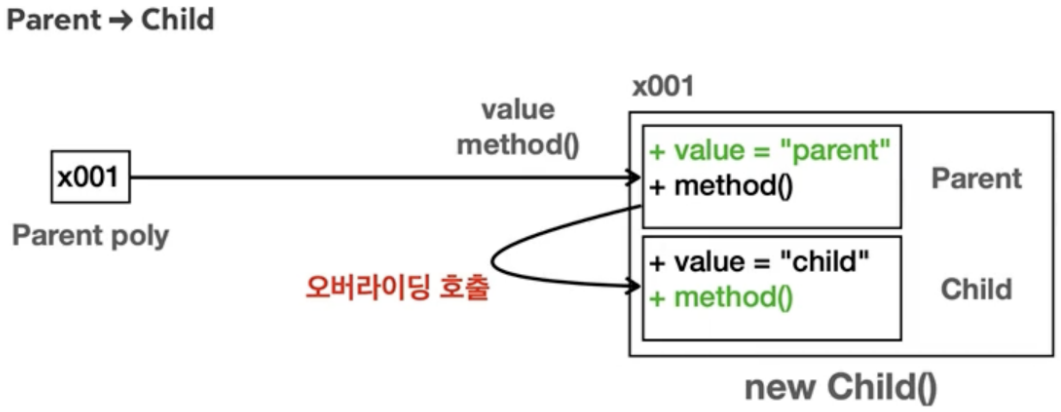
[다형성과 메서드 오버라이딩]

다형성을 이루는 또 하나의 중요한 핵심 이론.

오버라이딩 된 메서드가 항상 우선권을 가진다.

```
package poly.overriding;


public class OverridingMain {
    public static void main(String[] args) {
        //자식 변수가 자식 인스턴스 참조
        Child child = new Child();
        System.out.println("Child -> Child");
        System.out.println("value  = " + child.value);
        child.method();

        //부모 변수가 부모 인스턴스를 참조
        Parent parent = new Parent();
        System.out.println("Parent -> Parent");
        System.out.println("parent.value = " + parent.value);
        parent.method();

        //부모 변수가 자식 인스턴스를 참조
        Parent polyParent = new Child();
        System.out.println("Parent -> Child");
        System.out.println("polyChild.value = " + polyParent.
        polyParent.method(); //메서드는 오버라이딩 O
    }
}
==>
/Library/Java/JavaVirtualMachines/openjdk-17.jdk/Contents/Hom
Child -> Child
value  = child
child Method
Parent -> Parent
parent.value = parent
Parent Method
Parent -> Child
polyChild.value = parent
child Method

Process finished with exit code 0
```

- 변수는 Overriding이 되지 않아, parent가 출력되나,
  메서드는 Overriding되어 child Method 출력.



- Overriding된 메서드는 항상 우선권을 갖는다.