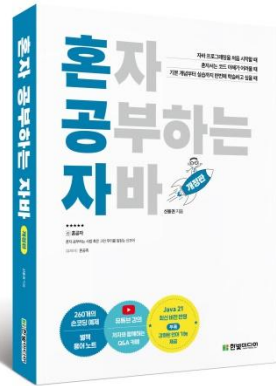


Chapter

07

상속



07-2. 타입 변환과 다형성

혼자 공부하는 자바(개정판) (신용권 저)

- 시작하기 전에
- 자동 타입 변환
- 필드의 다형성
- 매개변수의 다형성
- 강제 타입 변환
- 객체 타입 확인
- 키워드로 끝내는 핵심 포인트



시작하기 전에

[핵심 키워드] : 클래스 타입 변환, 자동 타입 변환, 다형성, 강제 타입 변환, instanceof

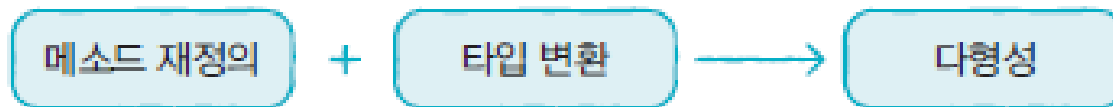
[핵심 포인트]

기본 타입과 마찬가지로 클래스도 타입 변환이 있다.

이를 활용하면 객체 지향 프로그래밍의 다형성을 구현할 수 있다.

❖ 다형성

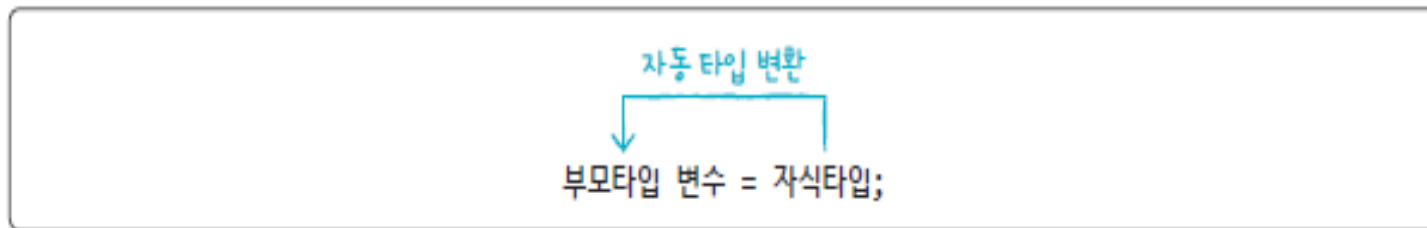
- 사용 방법은 동일하지만 다양한 객체 활용해 여러 실행결과가 나오도록 하는 성질
- 메소드 재정의와 타입 변환으로 구현



자동 타입 변환

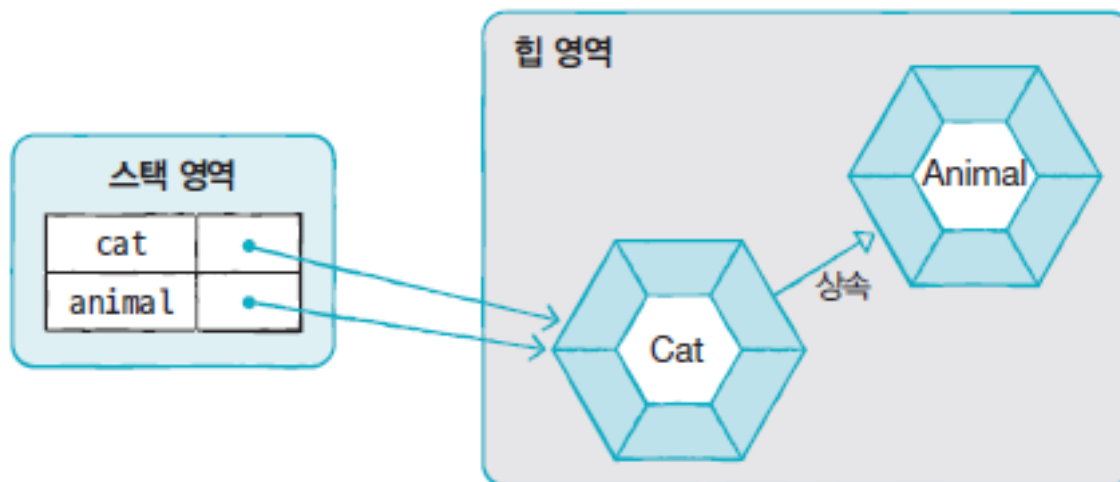
❖ 자동 타입 변환 (promotion)

- 프로그램 실행 도중 자동으로 타입 변환 일어나는 것



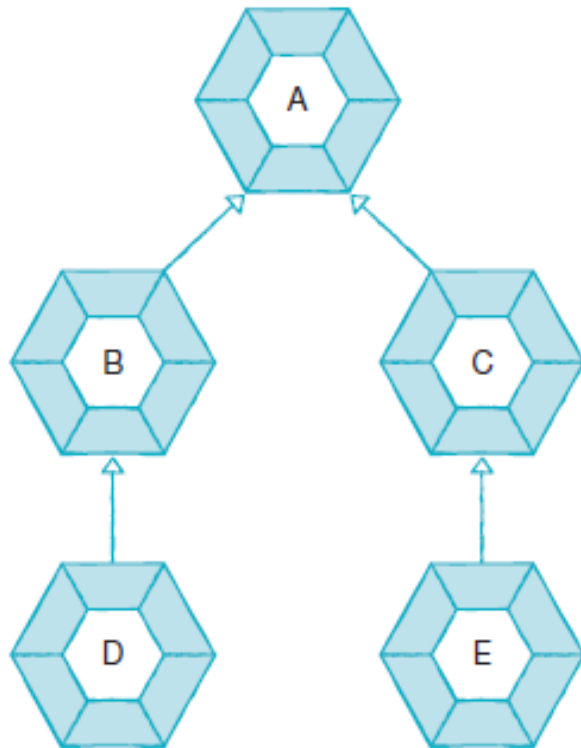
```
Cat cat = new Cat();  
Animal animal = cat;
```

← Animal animal = new Cat(); 도 가능



자동 타입 변환

- 바로 위 부모가 아니더라도 상속 계층에서 상위 타입인 경우 자동 타입 변환 일어날 수 있음



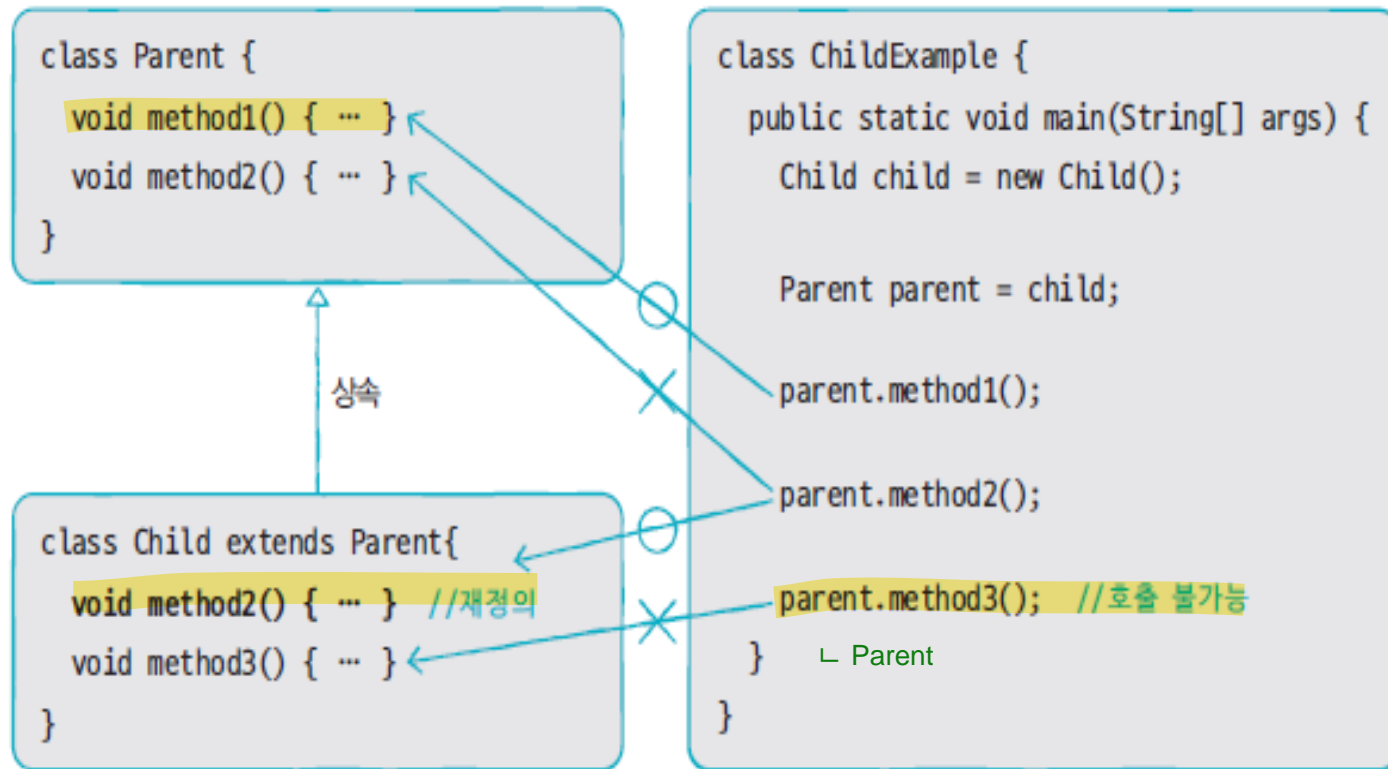
```
B b = new B( );  
C c = new C( );  
D d = new D( );  
E e = new E( );
```

```
A a1 = b; //(가능)  
A a2 = c; //(가능)  
A a3 = d; //(가능)  
A a4 = e; //(가능)  
  
B b1 = d; //(가능)  
C c1 = e; //(가능)  
  
B b3 = e; //(불가능)  
C c2 = d; //(불가능)
```



자동 타입 변환

- 부모 타입으로 자동 타입 변환 이후에는 부모 클래스에 선언된 필드 및 메소드만 접근 가능
- 예외적으로, 메소드가 자식 클래스에서 재정의될 경우 자식 클래스의 메소드가 대신 호출

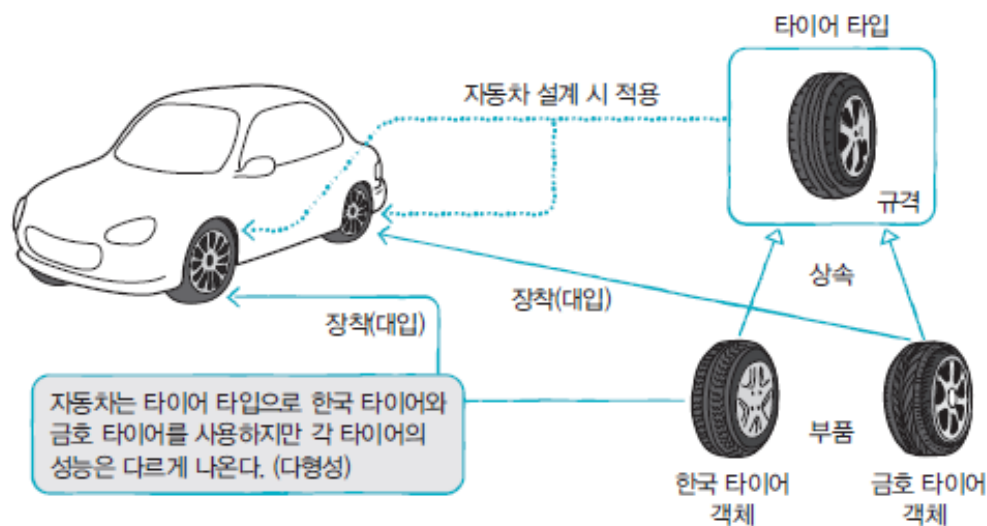


필드의 다형성

❖ 필드의 다형성

- 필드 타입을 부모 타입으로 선언할 경우
 - 다양한 자식 객체가 저장되어 필드 사용 결과 달라질 수 있음

```
class Car {  
    //필드  
    Tire frontLeftTire = new Tire();  
    Tire frontRightTire = new Tire();  
    Tire backLeftTire = new Tire();  
    Tire backRightTire = new Tire();  
    //메소드  
    void run() {  
        frontLeftTire.roll();  
        frontRightTire.roll();  
        backLeftTire.roll();  
        backRightTire.roll();  
    }  
}
```



```
Car myCar = new Car();  
myCar.frontRightTire = new HankookTire();  
myCar.backLeftTire = new KumhoTire();  
myCar.run();
```



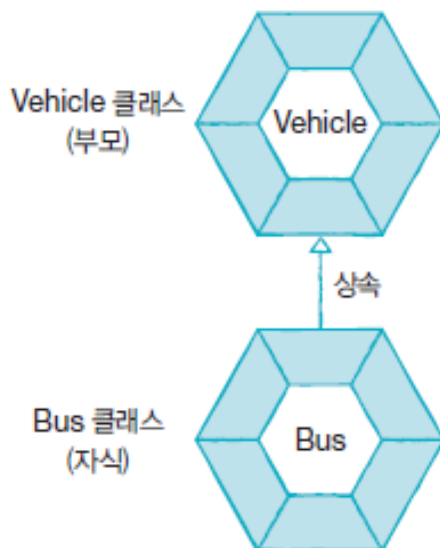
매개 변수의 다형성

❖ 매개 변수의 다형성

- 매개 변수를 부모 타입으로 선언하는 효과
 - 메소드 호출 시 매개값으로 부모 객체 및 모든 자식 객체를 제공할 수 있음
 - 자식의 재정의된 메소드가 호출 -> 다형성

```
class Driver {  
    void drive(Vehicle vehicle) {  
        vehicle.run();  
    }  
}
```

```
Driver driver = new Driver();  
Vehicle vehicle = new Vehicle();  
driver.drive(vehicle);
```



```
Driver driver = new Dirver();  
Bus bus = new Bus();  
driver.drive( bus );
```

자동 타입 변환 발생
Vehicle vehicle = bus;



강제 타입 변환

❖ 강제 타입 변환 (casting)

- 부모 타입을 자식 타입으로 변환
 - 조건: 자식 타입이 부모 타입으로 자동 타입 변환한 후 다시 반대로 변환할 때 사용

자식타입 변수 = (자식타입) 부모타입;
부모 타입을 자식 타입으로 변환

```
Parent parent = new Child(); //자동 타입 변환  
Child child = (Child) parent; //강제 타입 변환
```

```
class Parent {  
    String field1;  
    void method1() { ... }  
    void method2() { ... }  
}
```

상속

```
class Child extends Parent {  
    String field2;  
    void method3() { ... }  
}
```

```
class ChildExample {  
    public static void main(String[] args) {  
        Parent parent = new Child();  
        parent.field1 = "xxx";  
        parent.method1();  
        parent.method2();  
        parent.field2 = "yyy"; //불가능  
        parent.method3(); //불가능  
  
        Child child = (Child) parent;  
        child.field2 = "yyy"; //가능  
        child.method3(); //가능  
    }  
}
```

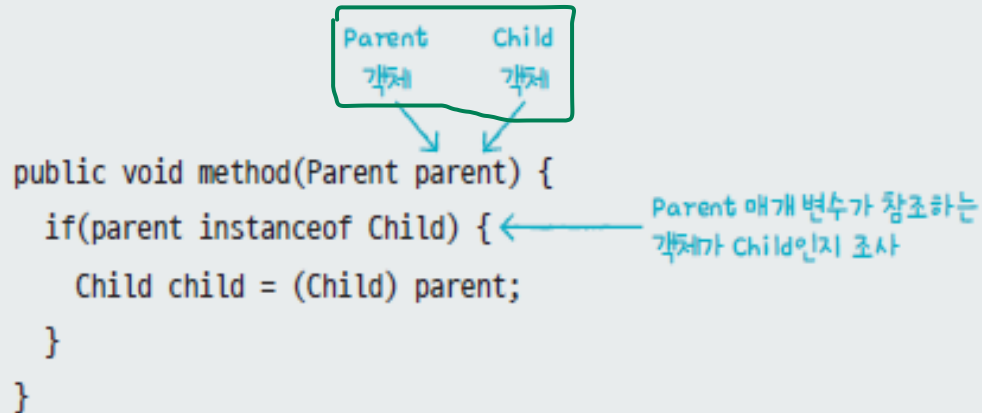
객체 타입 확인

❖ instanceof 연산자

- 어떤 객체가 어느 클래스의 인스턴스인지 확인
- 메소드 내 강제 타입 변환 필요한 경우
 - 타입 확인하지 않고 강제 타입 변환 시도 시 `ClassCastException` 발생할 수 있음
 - instanceof 연산자 통해 확인 후 안전하게 실행

```
boolean result = 좌항(객체) instanceof 우항(타입)
```

```
Parent parent = new Parent();  
Child child = (Child) parent; //강제 타입 변환을 할 수 없음
```



The diagram illustrates the use of the `instanceof` operator. At the top, two boxes labeled 'Parent 객체' and 'Child 객체' are shown. Arrows from both boxes point to the `parent` parameter in the `method` function signature of the code below. The code snippet is as follows:

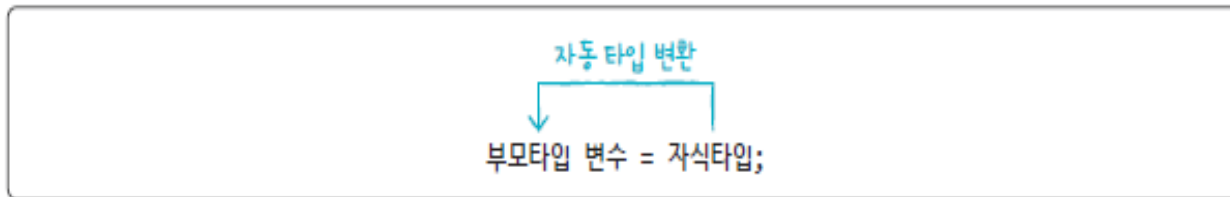
```
public void method(Parent parent) {  
    if(parent instanceof Child) {  
        Child child = (Child) parent;  
    }  
}
```

A blue arrow points from the text 'Parent 매개 변수가 참조하는 객체가 Child인지 조사' (Check if the object referenced by the Parent parameter is a Child) to the `parent instanceof Child` condition in the code.

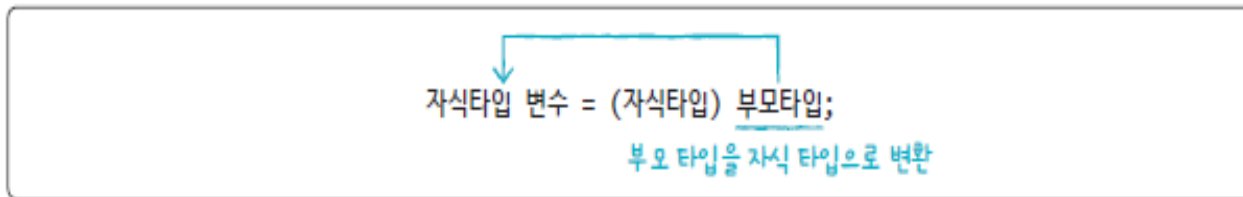


키워드로 끝내는 핵심 포인트

- **클래스 타입 변환** : 다른 클래스 타입으로 객체를 대입
- **자동 타입 변환** : 자식 객체를 부모 타입 변수에 대입할 때에는 자동으로 타입이 변환됨

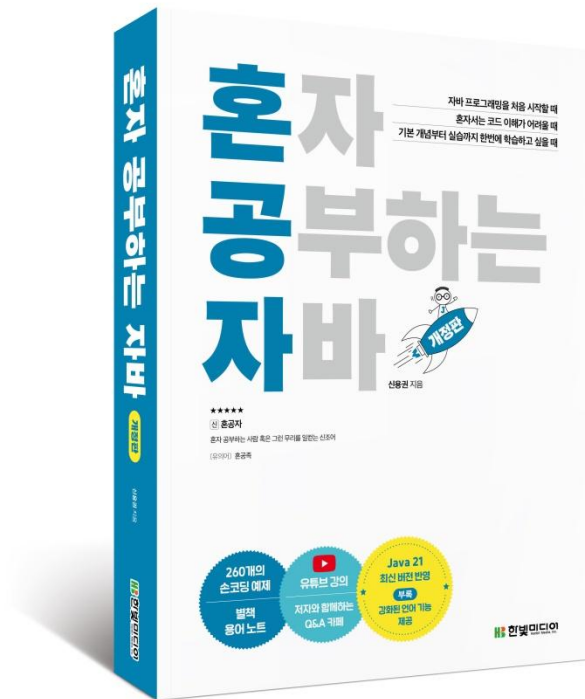


- **강제 타입 변환** : 부모 타입 객체를 다시 자식 타입에 대입할 때 강제 타입 변환일 필요



- **instanceof 연산자** : 객체가 어떤 타입인지 조사할 때 instanceof 연산자 사용.
- **다형성** : 객체 사용 방법은 동일하나 실행결과가 다양하게 나오는 성질.
메소드 재정의와 타입 변환으로 구현.





Thank You!