JAVA 기초입문과정

CHAPTER07 상속과 다형성

Contents

PART 01 상속

PART 02 클래스와 형변환

PART 03 메소드 오버라이드

PART 04 다형성

PART 05 다운캐스팅과 instanceof


PART
01
“
상속
”

*** 상속의 유형**

상속은 클래스, 추상클래스, 인터페이스를 통해 구현된다

1. 클래스 8장
2. 추상클래스 9장
3. 인터페이스 10장

PART 01

상속

1. 상속이란?

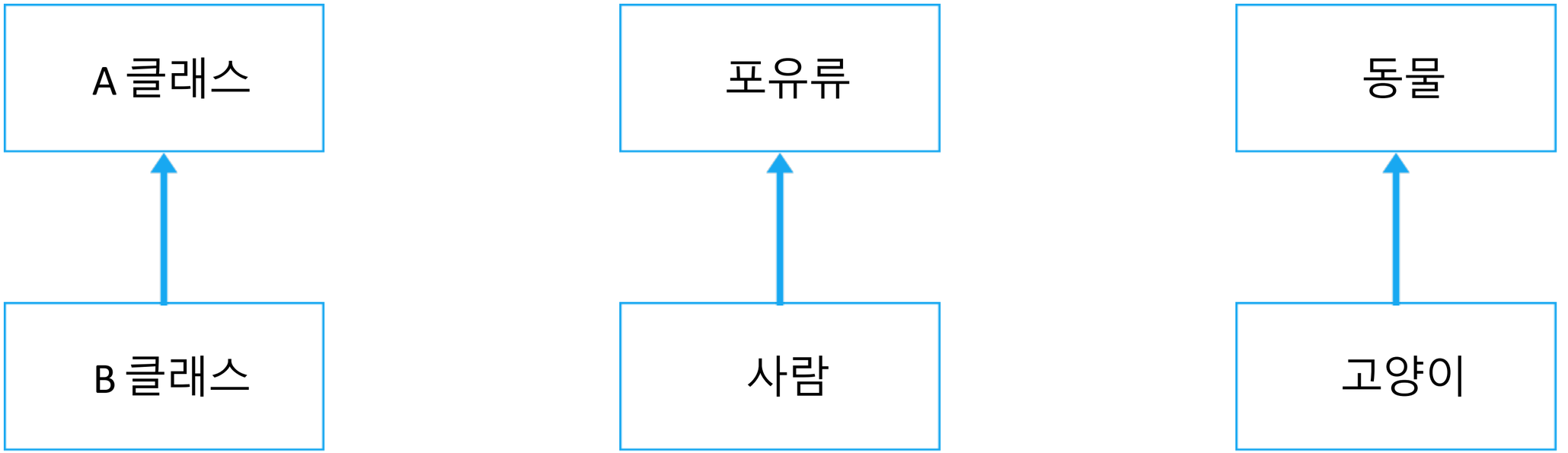
자식이 부모의 재산을 물려 받는것과 유사

부모가 자식에게 재산을 물려준다

자식은 상속받은 재산을
자신의 것으로 사용할수 있다

2. 클래스의 상속

아래와 같은 관계를 생각해보자!



B클래스가 A클래스를 상속받는다

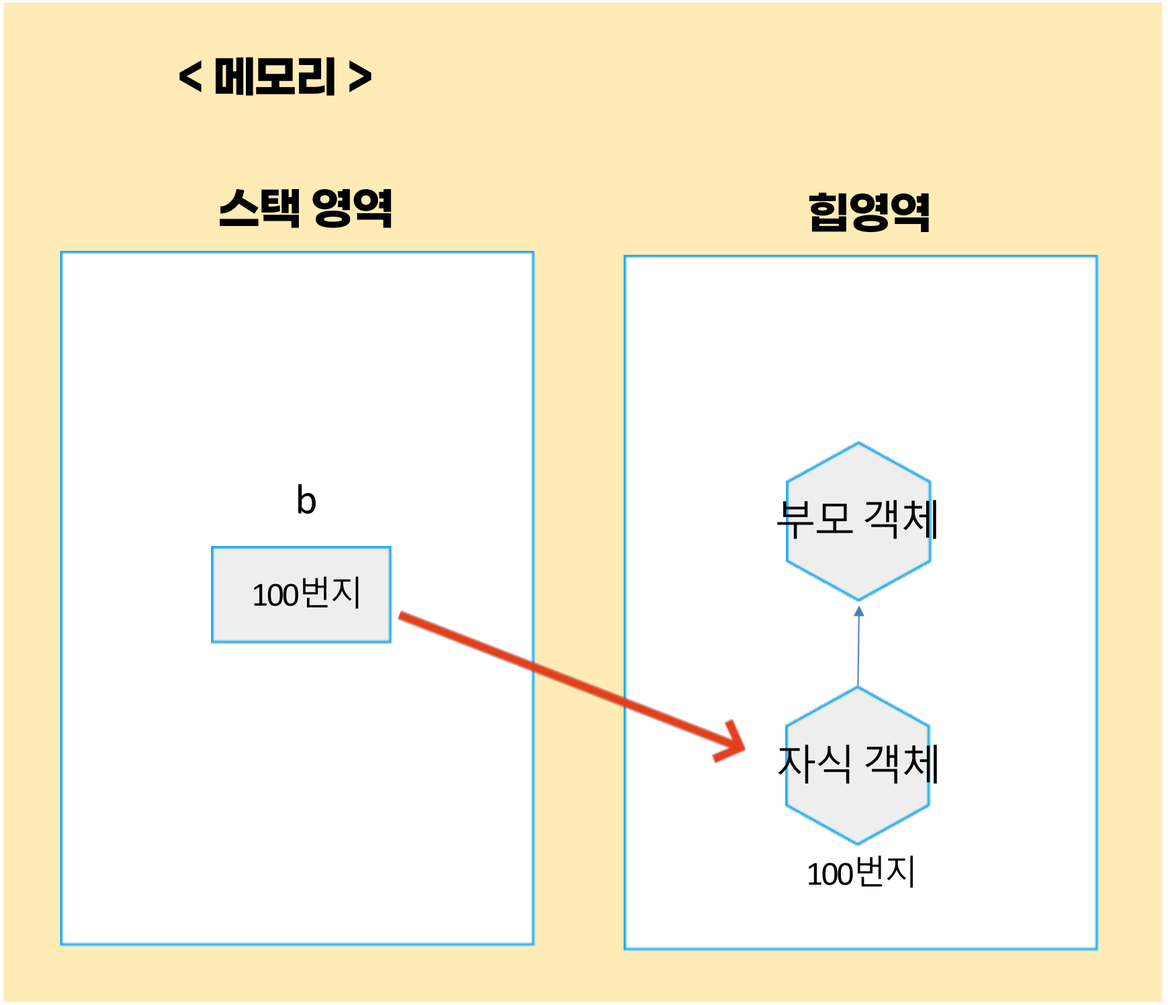
3. 상속 사용하기

부모클래스 선언

```
public class A {
}
```

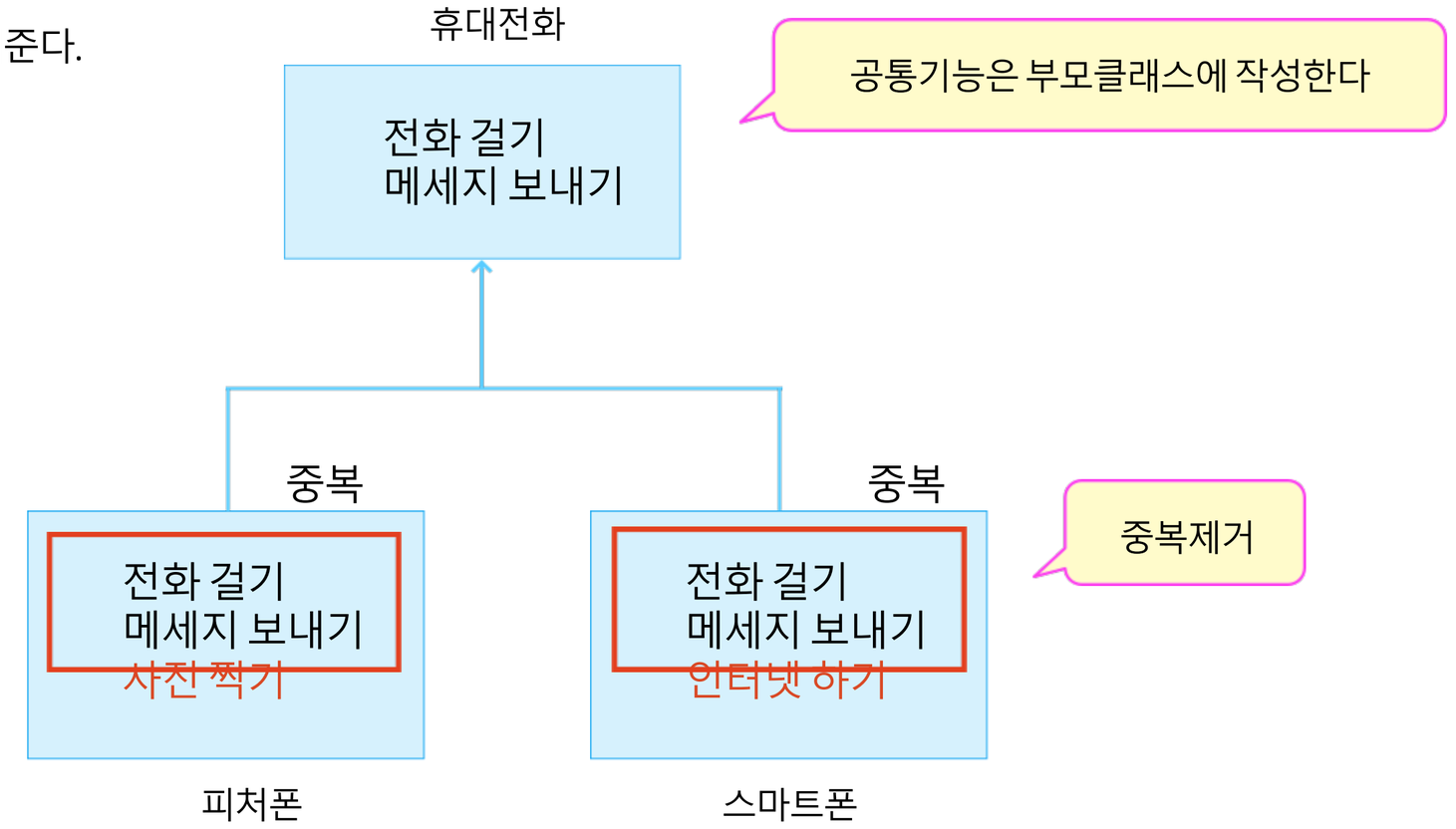
자식클래스 선언

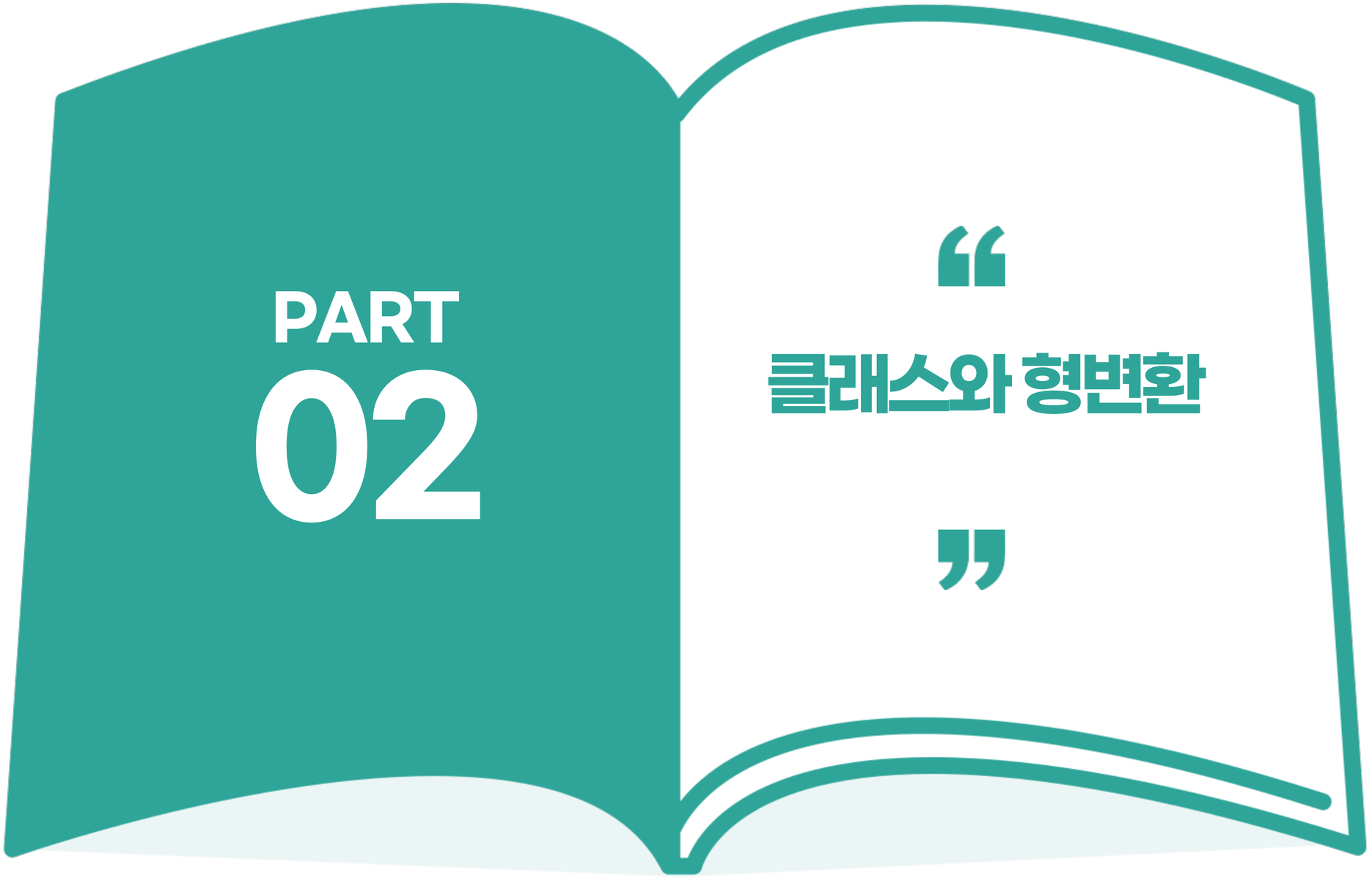
```
public class B extends A {
}
```



* 상속의 특징

- 부모의 필드와 메소드를 자식클래스에 물려준다.
- 부모의 클래스를 물려받아 확장한다.
- 중복코드가 제거된다.




PART
02
“
클래스와 형변환
”

1. 하위클래스가 생성되는 과정

* 상위클래스

```
public class Phone{
    public Phone ( ){
        출력 ( "Phone 생성자 실행" )
    }
}
```

1

* 하위클래스

```
public class SmartPhone extends Phone {
    public SmartPhone (String model, String color ){
        super ();
        출력 ( "SmartPhone 생성자 실행" )
    }
}
```

2

* SmartPhone 인스턴스 생성

[실행결과]
Phone생성자 실행
SmartPhone 생성자 실행

상위클래스의 생성자가 먼저 호출되고,
그다음에 하위클래스의 생성자가 호출된다

PART 02

메소드 오버라이드

1. 상위클래스 메소드 재정의하기

조건 - 단, 반환형, 메소드이름, 매개변수가 모두 같아야한다

- 상속된 메소드를 자식 클래스에서 재정의 하는 것.
- 어떤 메소도는 자식 클래스가 사용하기에 적합하지 않을 수 있다.
  이런 메소드는 자식 클래스에서 재정의(오버라이드) 해서 사용.

* 상위클래스

```
public class Calculator {
  //메소드 선언
  public double areaCircle(double r) {
      System.out.println("Calculator 객체의 areaCircle() 실행");
      return 3.14159 * r * r;
  }
}
```

* 하위클래스

```
public class Computer extends Calculator {
//메소드 오버라이딩
   @Override
    public double areaCircle(double r) {
       System.out.println("Computer 객체의 areaCircle() 실행");
       return Math.PI * r * r;
    }
}
```

재정의한 메소드

* 이클립스의 generate 기능을 사용하자

2. super

- 부모를 가리키는 예약어

언제사용할까? 상위 클래스의 생성자를 호출하때

* 상위클래스

```
public class Phone{
   public Phone ( ){
      출력 ( ”Phone 생성자 실행" )
   }
}
```

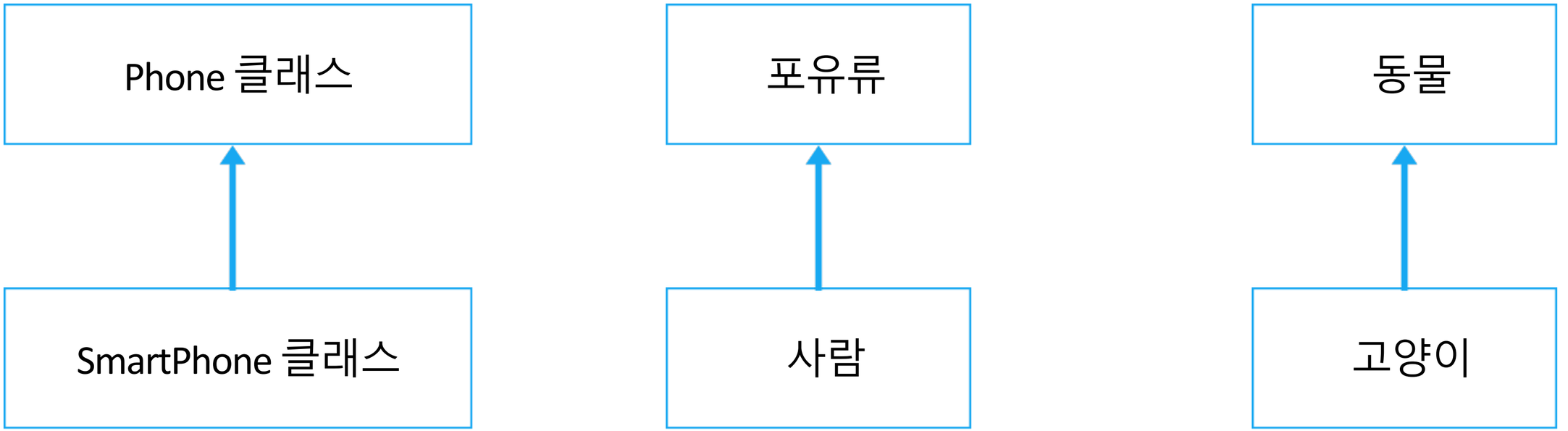
* 하위클래스

```
public class SmartPhone extends Phone {
    public SmartPhone (String model, String color ){
        super ();
        출력 ( ”SmartPhone 상
    }
}
```

상위클래스 생성자 호출
(매개 변수가 없다면 생략가능)

3. 상위클래스로 형변환하기

다음의 관계들을 생각해보자!



SmartPhone 클래스는 Phone 클래스이다???

사람은 포유류이다

고양이는 동물이다

PART 02

클래스와 형변환

- **자동 타입 변환**

1. 부모 클래스에 선언된 필드와 메소드에만 접근이 가능하다.
2. 자식 클래스에 오버라이딩 된 메소드가 있다면 오버라이딩 된 메소드가 호출.

```
Child child = new Child();
Parent parent = child;
```

자식타입 인스턴스 생성

부모타입으로 자동 형변환!

- **강제 타입 변환**

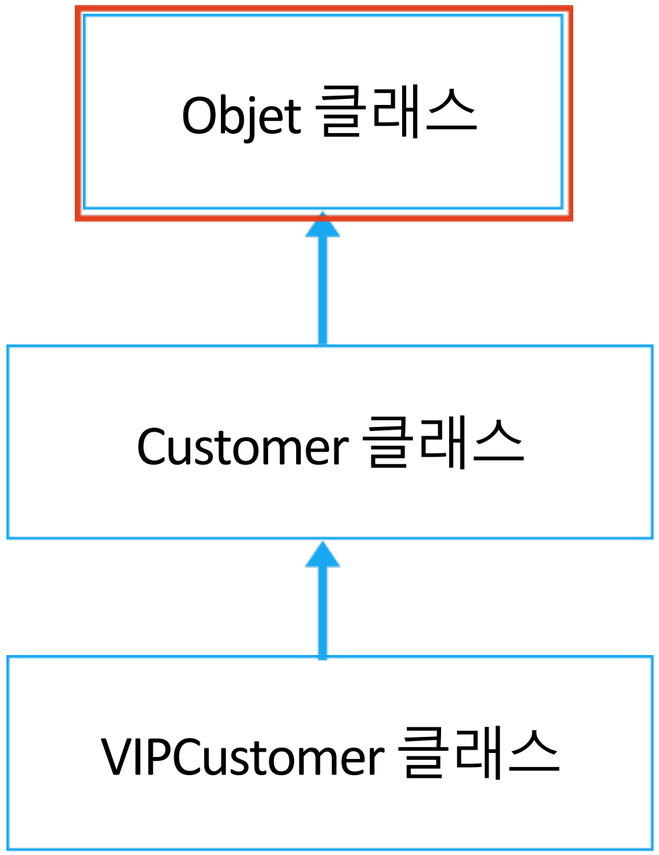
1. 부모타입에 선언된 필드와 메소드만 사용 가능.
2. 만약 자식 타입에 선언된 필드와 메소드를 사용해야 한다면 강제 타입 변환.

```
Parent parent = new Child();
Child child = (child)parent;
```

부모 변수로 받는 자식 인스턴스 생성

자식타입으로 강제 형변환!

* 최상위 클래스 Object



모든클래스는 Object 클래스를 상속받는다


PART
03
“
메소드 오버라이드
”

PART 03

메소드 오버라이드

*** 자동형변환 + 메소드 재정의를 동시에 했을경우, 어떤 메소드가 호출될까?**

형변환

```
Customer vc = new VIPCustomer();

vc.calcPrice();
```

```
public class Customer {
    public int calcPrice ( int price ) {
        return price;
    }
}
public class VIPCustomer {
    @Override
    public int calcPrice ( int price ) {
        return price - (int) (price - price*saleRatio )
    }
}
```

재정의된 메소드

-> 부모 타입의 자료형이더라도,
메소드 이름이 같을때는 재정의된 메소드를 호출한다

PART
04

“
다형성
”

PART 04

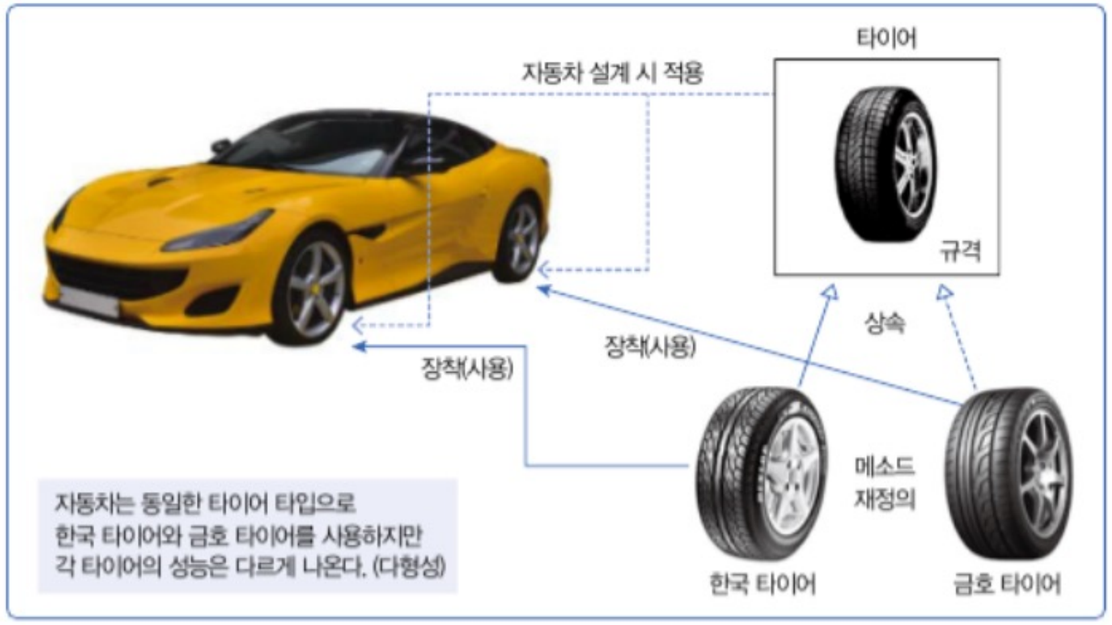
클래스 다형성

1. 다형성

형변환과 메소드 오버라이드를 바탕으로 객체지향의 다형성을 이해해보자!

클래스 다형성

1. 다형성

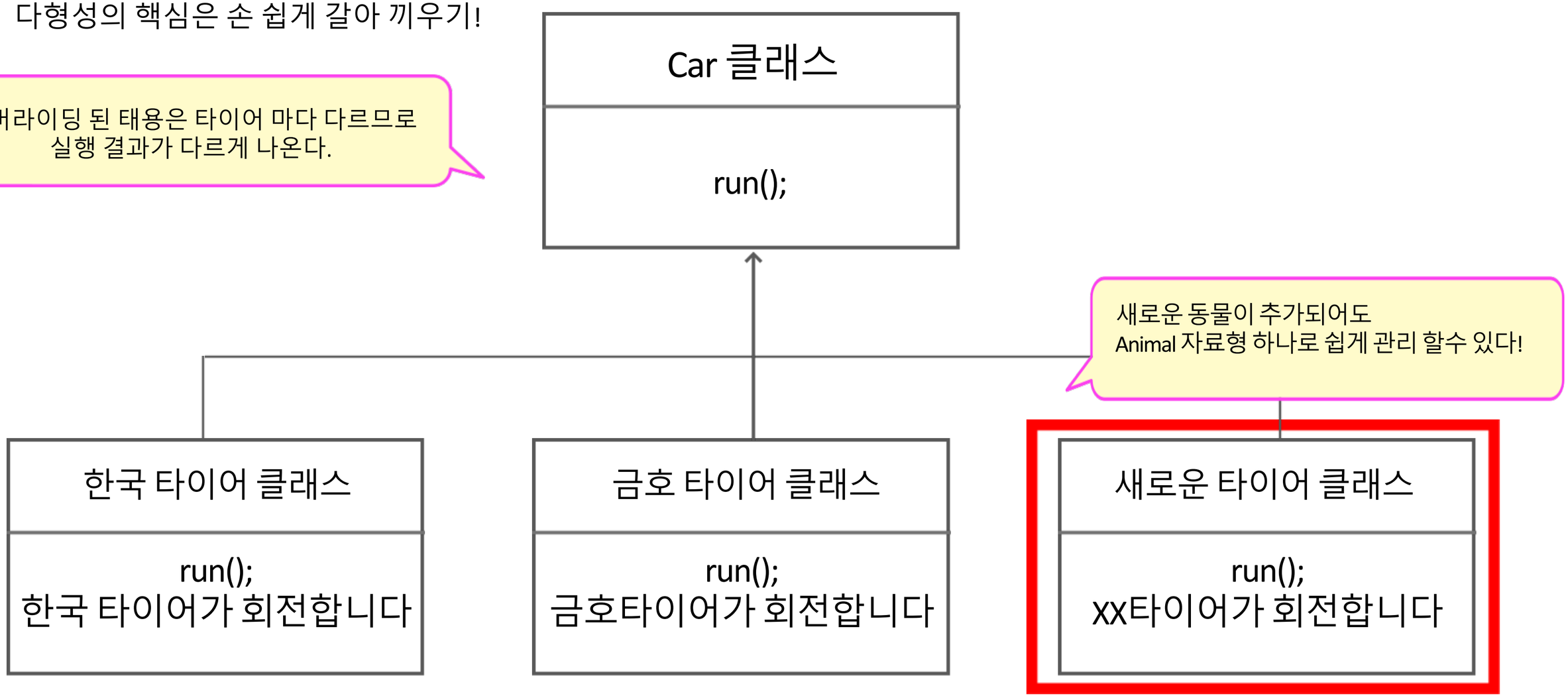

타이어
자동차 설계 시 적용
규격
상속
장착(사용)
장착(사용)
메소드
재정의
자동차는 동일한 타이어 타입으로
한국 타이어와 금호 타이어를 사용하지만
각 타이어의 성능은 다르게 나온다. (다형성)
한국 타이어
금호 타이어


Tire
roll()
규격
상속
roll()
재정의
HankookTire
KumhoTire

2. 필드 다형성

- 다형성의 핵심은 손 쉽게 갈아 끼우기!

오버라이딩 된 태용은 타이어 마다 다르므로
실행 결과가 다르게 나온다.

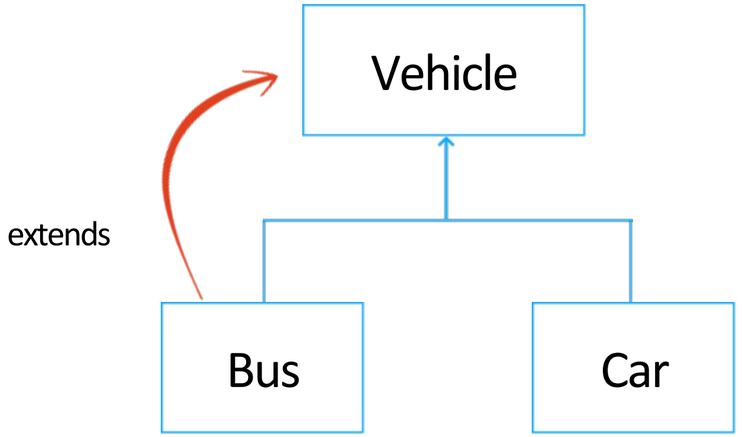


2. 매개변수 다형성

메소드 매개변수

```
Driver driver = new Driver();

Bus bus = new Bus();
driver.drive(bus);
Taxi taxi = new Taxi();
driver.drive(taxi);
```



호출할때는 자식 객체를 대입

Bus Taxi

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

매개변수를 부모타입으로 선언

-> Vehicle의 모든 자식 클래스를 받을 수 있다.


PART
05
“
instanceof
”

PART 05

instanceof

- instanceof : 변수가 참조하는 객체의 타입을 확인하고자 할 때

강제 타입변환을 하는 이유는
Student의 모든 멤버(필드, 메소드)에
접근하기 위해

```
if (person instanceof Student) {
    //Student 객체일 경우 강제 타입 변환
    Student student = (Student) person;

    //Student 객체만 가지고 있는 필드 및 메소드 사용
     System.out.println("studentNo: " + student.studentNo);
     student.study();
 }
```

Thank You!