Chapter 07

상속

07-1. 상속

혼자 공부하는 자바 (신용권 저)

한빛미디어 Hanbit Media, Inc.

목차

시작하기 전에

[핵심 키워드] : 상속, 메소드 재정의, final 클래스, final 메소드

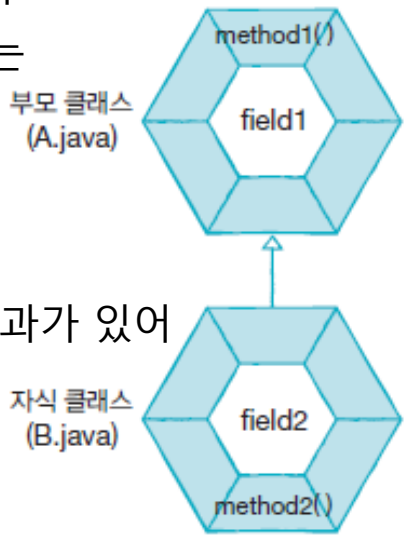
[핵심 포인트]

객체 지향 프로그램에서 부모 클래스의 멤버를 자식 클래스에게 물려줄 수 있다.

❖ 상속

- 이미 개발된 클래스를 재사용하여 새로운 클래스를 만들기에 중복되는 코드를 줄임

- 부모 클래스의 한번의 수정으로 모든 자식 클래스까지 수정되는 효과가 있어 유지보수 시간이 줄어듬




```
public class A {
  int field1;
  void method1() { … }
}
```

A를 상속

```
public class B extends A {
  String field2;
  void method2() { … }
}
```

클래스 상속

❖ 클래스 상속

- 자식 클래스 선언 시 부모 클래스 선택
- extends 뒤에 부모 클래스 기술

```
class 자식클래스 extends 부모클래스 {
  //필드
  //생성자
  //메소드
}
```

```
class SportsCar extends Car {
}
```

- 여러 개의 부모 클래스 상속할 수 없음
- 부모 클래스에서 private 접근 제한 갖는 필드와 메소드는 상속 대상에서 제외
- 부모와 자식 클래스가 다른 패키지에 존재할 경우 default 접근 제한된 필드와 메소드 역시 제외

부모 생성자 호출

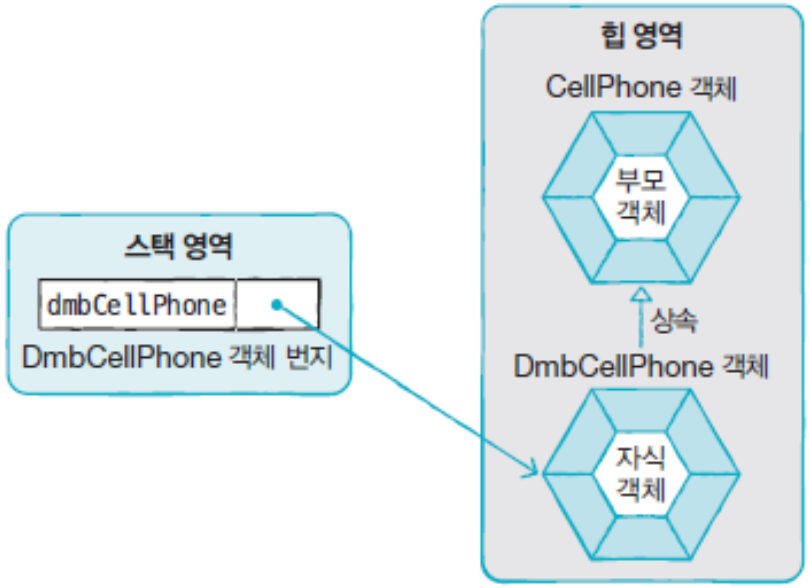
❖ 자식 객체 생성할 때 부모 객체가 먼저 생성되고 그 다음 자식 객체가 생성됨

```
DmbCellPhone dmbCellPhone = new DmbCellPhone();
```

- 자식 생성자의 맨 첫 줄에서 부모 생성자가 호

```
public DmbCellPhone() {
  super();
}
```

```
public CellPhone() {
}
```



- 명시적으로 부모 생성자 호출하려는 경우

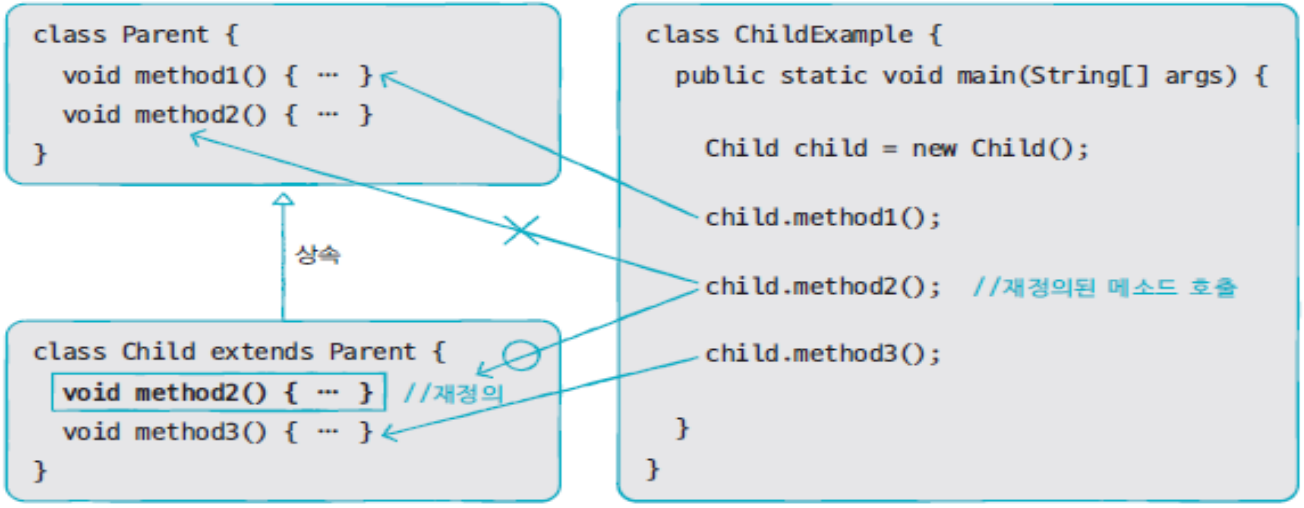
```
자식클래스( 매개변수선언, … ) {
  super( 매개값, … );
  …
}
```

메소드 재정의

❖ 메소드 재정의 (오버라이딩 / Overriding)

- 부모 클래스의 메소드가 자식 클래스에서 사용하기에 부적합할 경우 자식 클래스에서 수정하여 사용
- 메소드 재정의 방법
  - 부모 메소드와 동일한 시그니처 가져야 함
  - 접근 제한 더 강하게 재정의할 수 없음
  - 새로운 예외를 throws 할 수 없음
- 메소드가 재정의될 경우 부모 객체 메소드가 숨겨지며,

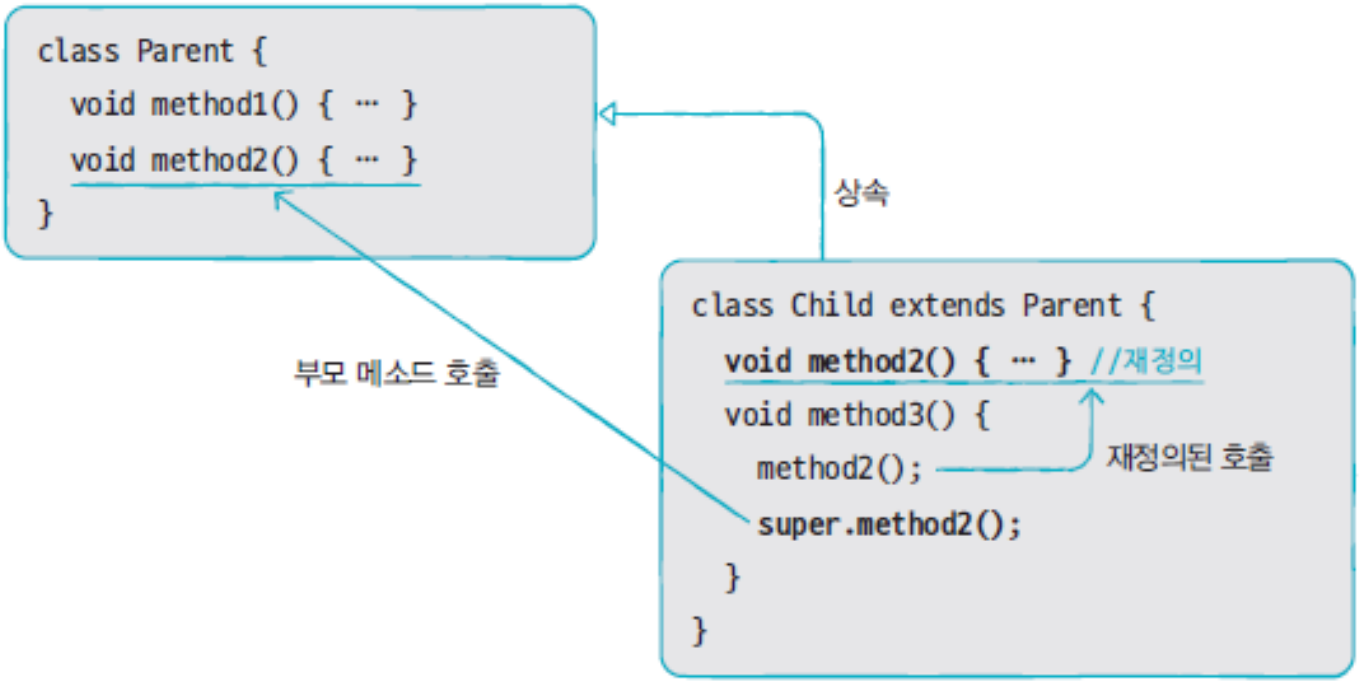
  자식 객체에서 메소드 호출하면 재정의된 자식 메소드가 호출됨



메소드 재정의

- 부모 메소드 호출
  - 자식 클래스 내부에서 재정의된 부모 클래스 메소드 호출해야 하는 경우
  - 명시적으로 super 키워드 붙여 부모 메소드 호출

```
super.부모메소드();
```



final 클래스와 final 메소드

❖ final 키워드

- 해당 선언이 최종 상태이며 수정될 수 없음을 의미
- 클래스 및 메소드 선언 시 final 키워드를 사용하면 상속과 관련됨

❖ 상속할 수 없는 final 클래스

- 부모 클래스가 될 수 없어 자식 클래스 만들 수 없음을 의미

```
public final class 클래스 { … }
```

```
public final class String { … }
```

```
public class NewString extends String { … }
```

❖ 재정의할 수 없는 final 메소드

- 부모 클래스에 선언된 final 메소드는 자식 클래스에서 재정의 할 수 없음

```
public final 리턴타입 메소드( [매개변수, …] ) { … }
```

키워드로 끝내는 핵심 포인트

- 상속: 부모 클래스의 필드와 메소드를 자식 클래스에서 사용할 수 있도록 한다.
- 메소드 재정의 : 부모 메소드를 자식 클래스에서 다시 정의하는 것을 의미한다.
- final 클래스 : final 클래스는 부모 클래스로 사용할 수 없다.
- final 메소드 : 자식 클래스에서 재정의할 수 없는 메소드이다.

Chapter 07

상속

07-2. 타입 변환과 다형성

혼자 공부하는 자바 (신용권 저)

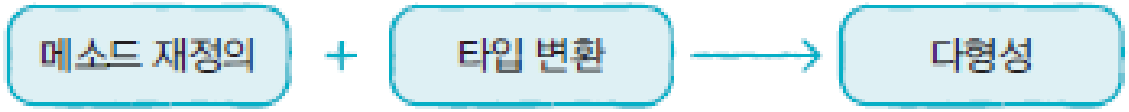
목차

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.

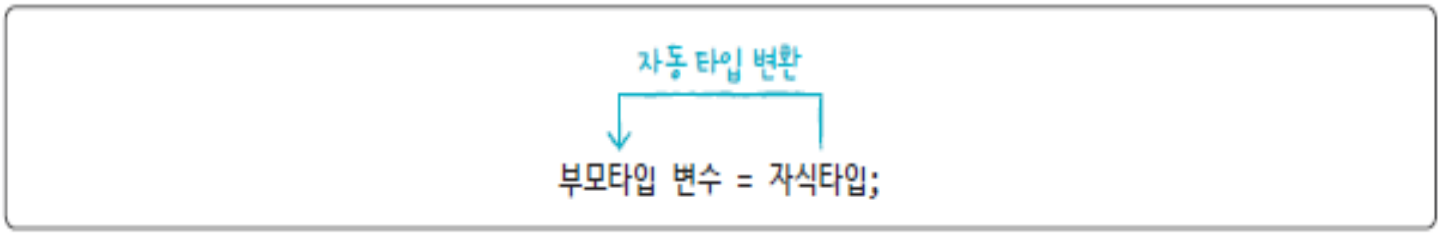
❖ 다형성
- 사용 방법은 동일하지만 다양한 객체 활용해 여러 실행결과가 나오도록 하는 성질
- 메소드 재정의와 타입 변환으로 구현



자동 타입 변환

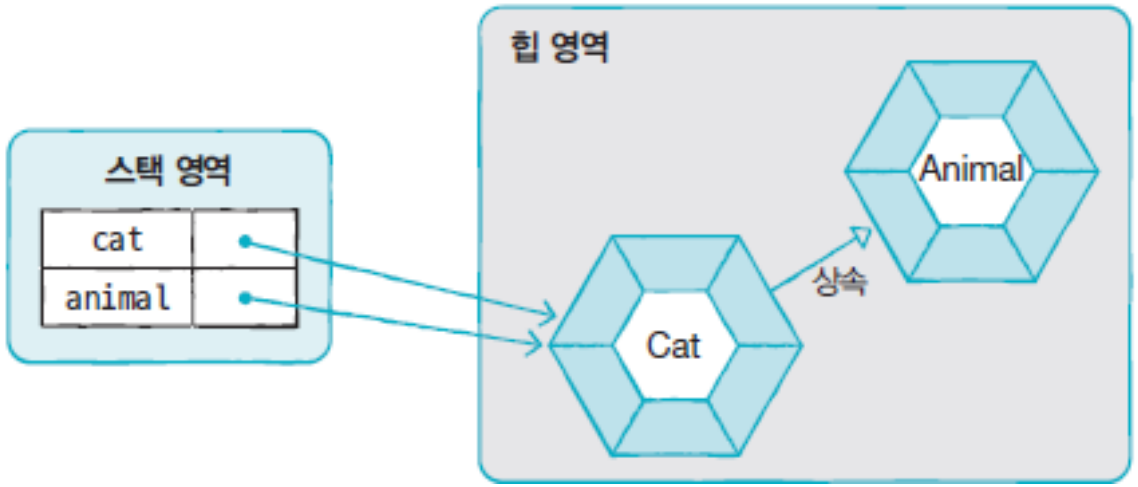
❖ 자동 타입 변환 (promotion)

- 프로그램 실행 도중 자동으로 타입 변환 일어나는 것



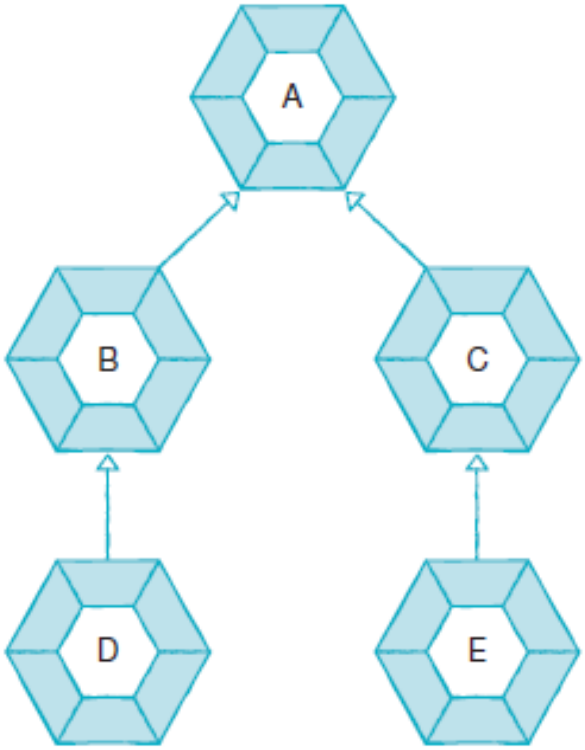

```
Cat cat = new Cat();
Animal animal = cat;
```

← 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( );
```

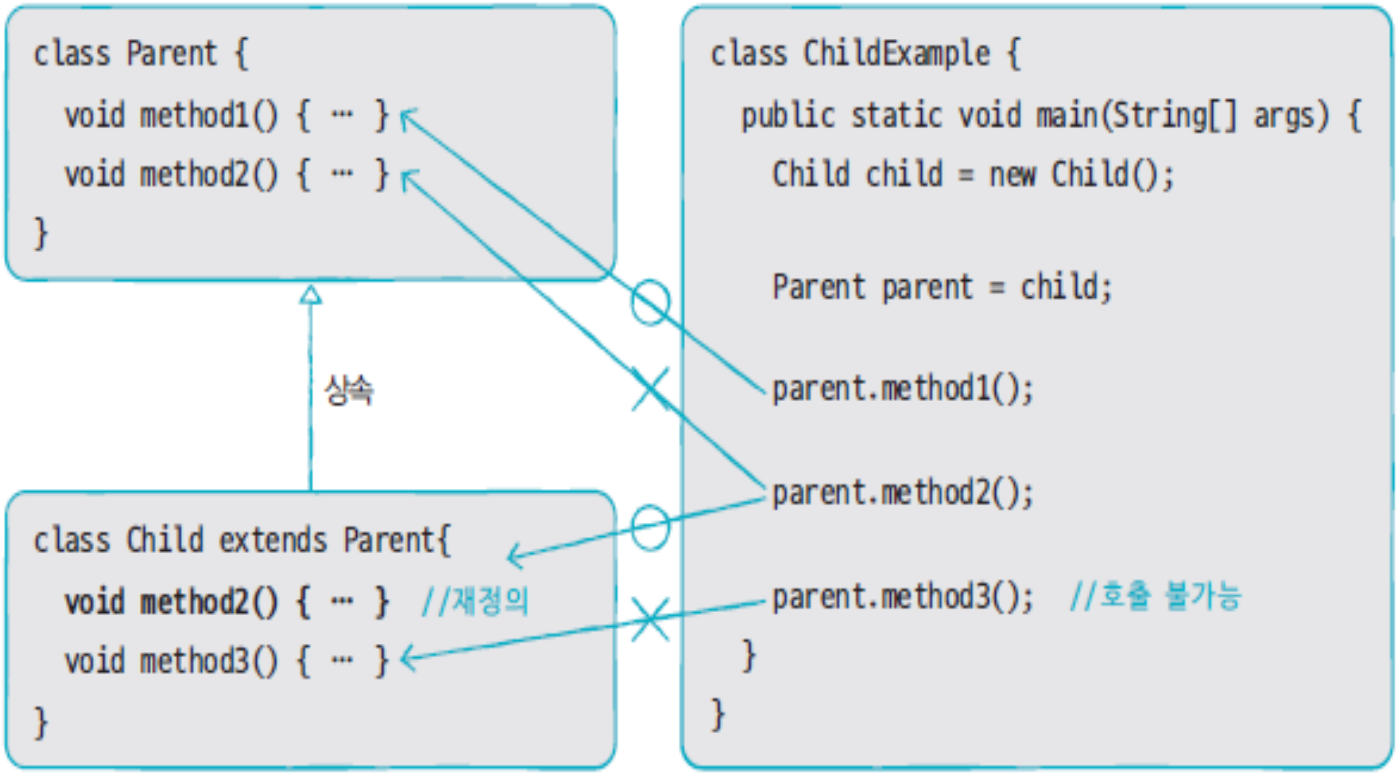
```
A a1 = b;  //(가능)
A a2 = c;  //(가능)
A a3 = d;  //(가능)
A a4 = e;  //(가능)

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

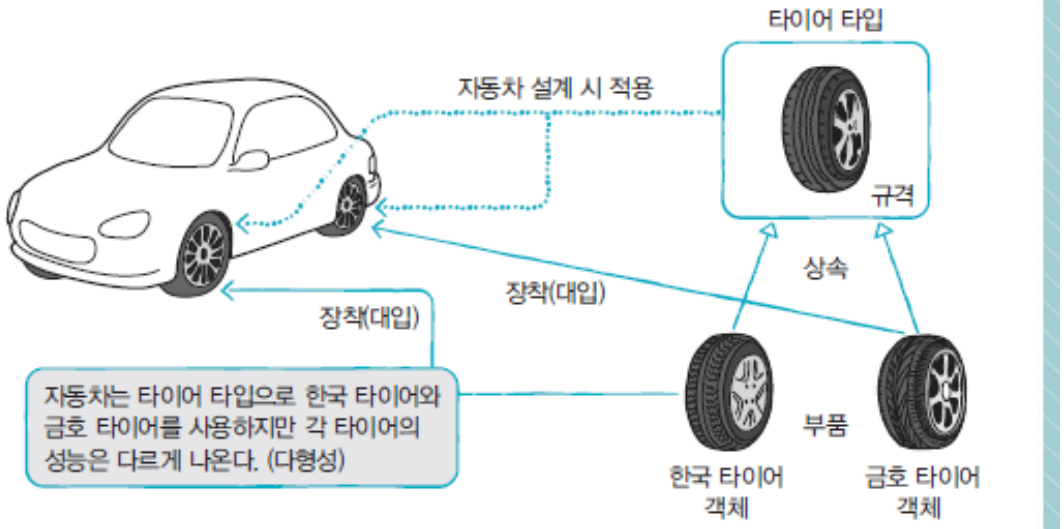


필드의 다형성

❖ 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();
```

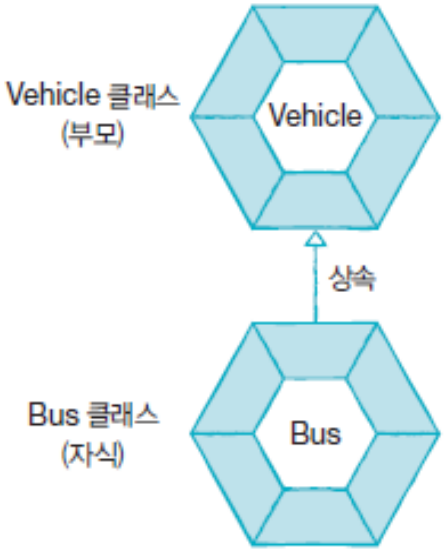
매개 변수의 다형성

❖ 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)
```

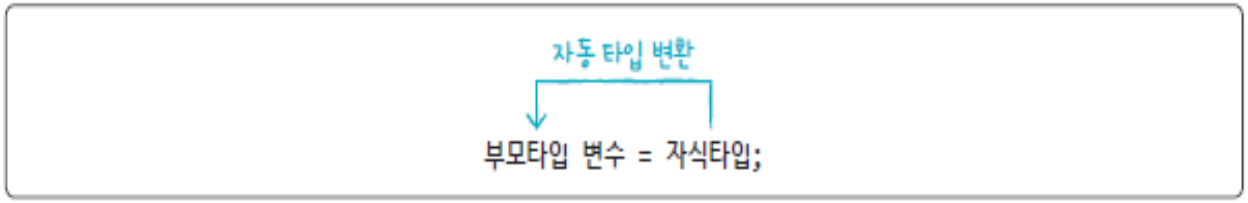
```
Parent parent = new Parent();
Child child = (Child) parent;     //강제 타입 변환을 할 수 없음
```

```
                    Parent      Child
                    객체        객체
                      ↘          ↙
public void method(Parent parent) {
  if(parent instanceof Child) {  ←——— Parent 매개 변수가 참조하는
                                      객체가 Child인지 조사
    Child child = (Child) parent;
  }
}
```

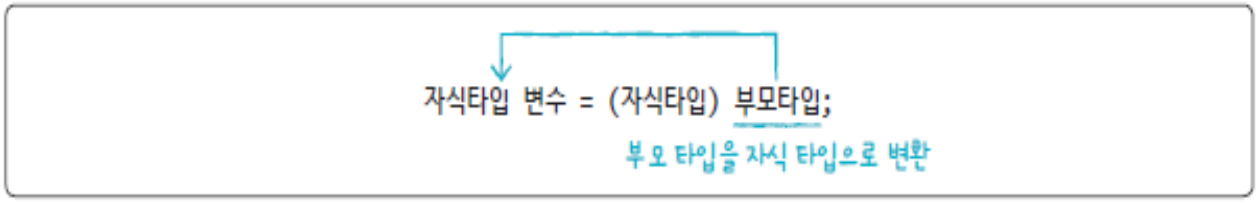
키워드로 끝내는 핵심 포인트

- 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.

Chapter 07

상속

07-3. 추상 클래스

혼자 공부하는 자바 (신용권 저)

목차

- 시작하기 전에
- 추상 클래스의 용도
- 추상 클래스 선언
- 추상 메소드와 재정의
- 키워드로 끝내는 핵심 포인트

시작하기 전에

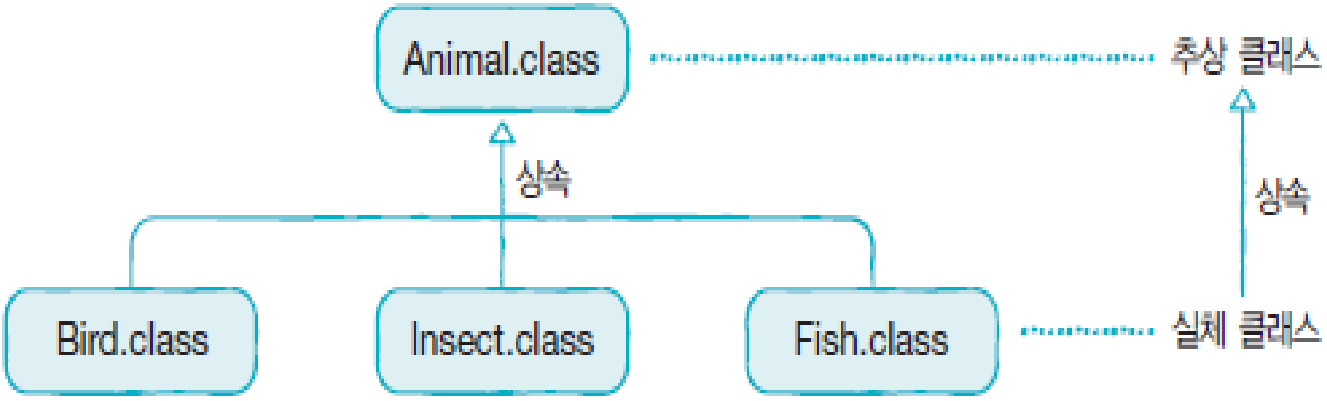
[핵심 키워드] : 추상 클래스, 추상 메소드, 재정의

[핵심 포인트]

여러 클래스의 공통된 특성(필드, 메소드)를 추출해서 선언한 것을 추상 클래스라고 한다.

❖ 추상 클래스

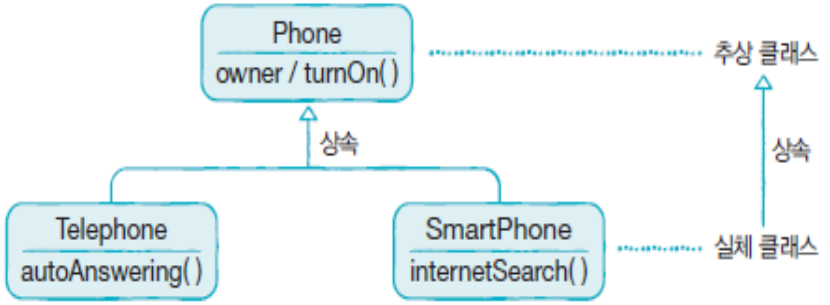
- 실체 클래스(객체 생성용 클래스)들의 공통적인 특성(필드, 메소드)을 추출하여 선언한 것
- 추상 클래스와 실체 클래스는 부모, 자식 클래스로서 상속 관계를 가짐

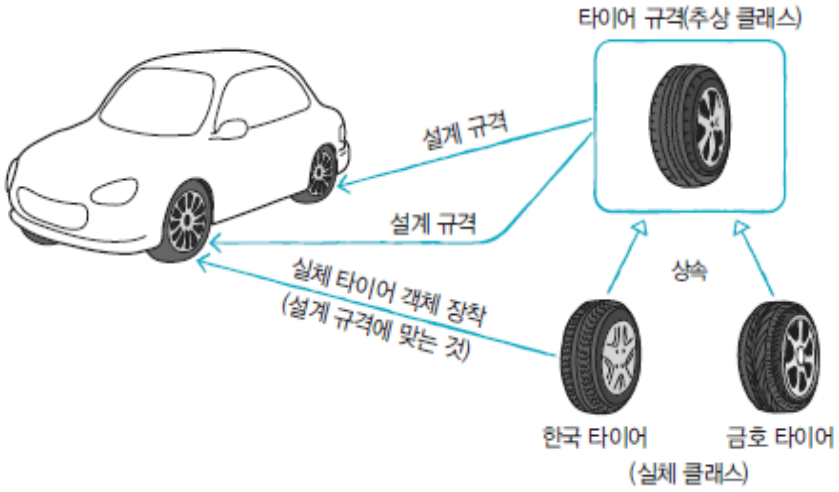


추상 클래스의 용도

❖ 추상 클래스의 용도

- 실체 클래스에 반드시 존재해야할 필드와 메소드의 선언(실체 클래스의 설계 규격 - 객체 생성 용이 아님)
- 실체 클래스에는 공통된 내용은 빠르게 물려받고 다른 점만 선언하면 되므로 시간 절약





추상 클래스 선언

❖ 추상 클래스 선언

- abstract 키워드
  - 상속 통해 자식 클래스만 만들 수 있게 만듬(부모로서의 역할만 수행)

```
public abstract class 클래스 {
  //필드
  //생성자
  //메소드
}
```

- 추상 클래스도 일반 클래스와 마찬가지로 필드, 생성자, 메소드 선언 할 수 있음

- 직접 객체를 생성할 수 없지만 자식 객체 생성될 때 객체화 됨.
  - 자식 생성자에서 super(...) 형태로 추상 클래스의 생성자 호출

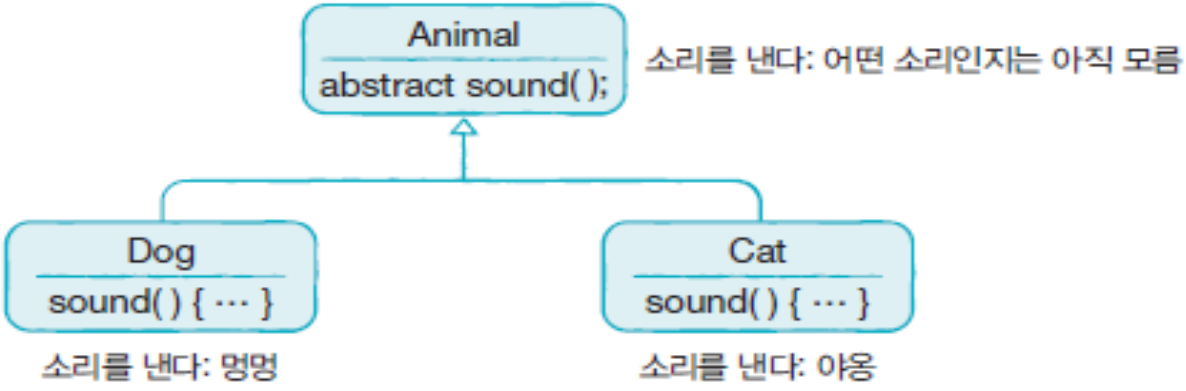
추상 메소드와 재정의

❖ 추상 메소드

- 메소드 선언만 통일하고 실행 내용은 실체 클래스마다 달라야 하는 경우
- abstract 키워드로 선언되고 중괄호가 없는 메소드
- 하위 클래스는 반드시 재정의해서 실행 내용을 채워야 함.

```
[public | protected] abstract 리턴타입 메소드이름(매개변수, …);
```

```
public abstract class Animal {
  public abstract void sound();
}
```



키워드로 끝내는 핵심 포인트

- 추상 클래스: 클래스들의 공통적인 필드와 메소드 추출하여 선언한 클래스

- 추상 메소드 :
  - 추상 클래스에서만 선언할 수 있고, 메소드의 선언부만 있는 메소드.
  - 자식 클래스에서 재정의되어 실행 내용 결정해야 함

Thank You!

혼자 공부하는 자바 (신용권 저)

한빛미디어 Hanbit Media, Inc.