7장. 상속

한림대학교 소프트웨어융합대학 양은샘.

7장. 상속

- 안녕하세요? 여러분!
- 오늘은 자바의 상속 단원을 학습 합니다.
- 이번 장에서는
  - 상속의 개념
  - 상속관계에서의 필드, 생성자, 메소드의 접근과 사용에 대해 알아보도록 하겠습니다.
- 지난 시간에 학습한 내용을 리뷰한 후 학습을 시작하도록 하겠습니다.

지난 시간 Review

- 객체 지향 프로그래밍
- 객체(Object)와 클래스(Class)
- 클래스 선언
- 객체 생성과 클래스 변수
- 클래스의 구성 멤버
- 필드(Field)
- 생성자(Constructor)
- 메소드(Method)
- 인스턴스 멤버와 this
- 정적 멤버와 static
- final 필드와 상수(static final)
- 패키지(package)
- 접근 제한자

학습 목차

1절. 상속 개념
2절. 클래스 상속(extends)

3절. 부모 생성자 호출(super(...))
4절. 메소드 재정의(Override)
5절. final 클래스와 final 메소드

6절. protected 접근 제한자

7절. 타입변환과 다형성(polymorphism)

8절. 추상 클래스(abstract Class)

학습 목표

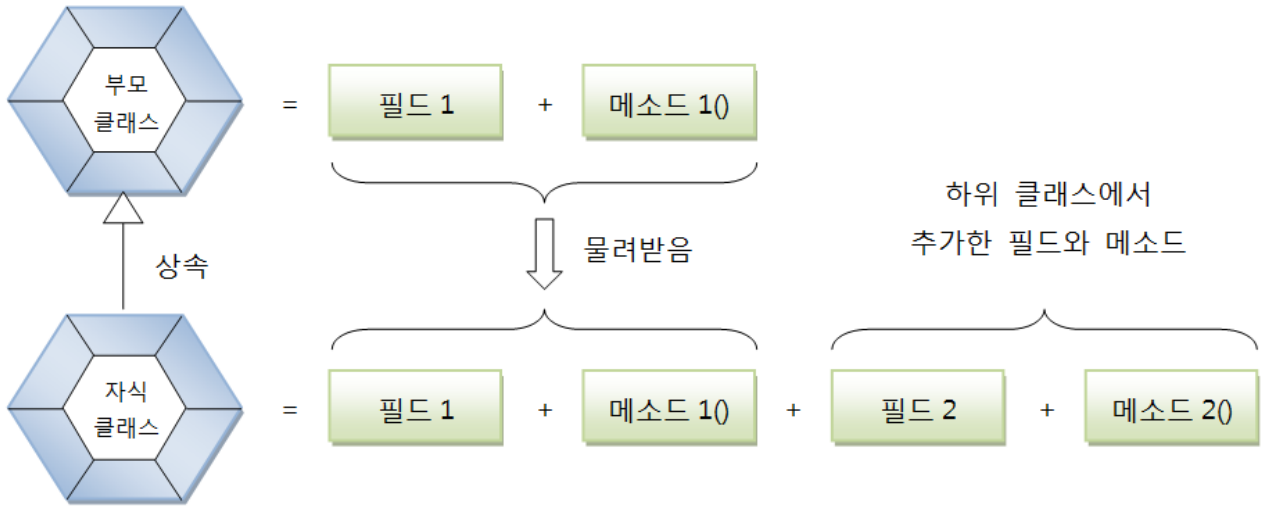
- 상속을 이용한 클래스를 작성할 수 있다.
- 상속된 클래스간의 상호 작용을 안다.
- 메소드를 재정의해서 사용할 수 있다.
- super. 과 super()의 사용법을 안다.
- 다형성의 개념을 이해하며 응용할 수 있다.
- 추상 클래스의 의미와 사용법을 안다.

상속(Inheritance)이란?

- 객체 지향 프로그램에서 상속(Inheritance)이란?

- 자식(하위, 파생) 클래스가 부모(상위) 클래스의 멤버를 물려받는 것
- 자식이 부모를 선택해 물려 받음

- 상속 대상
  - 부모의 필드와 메소드



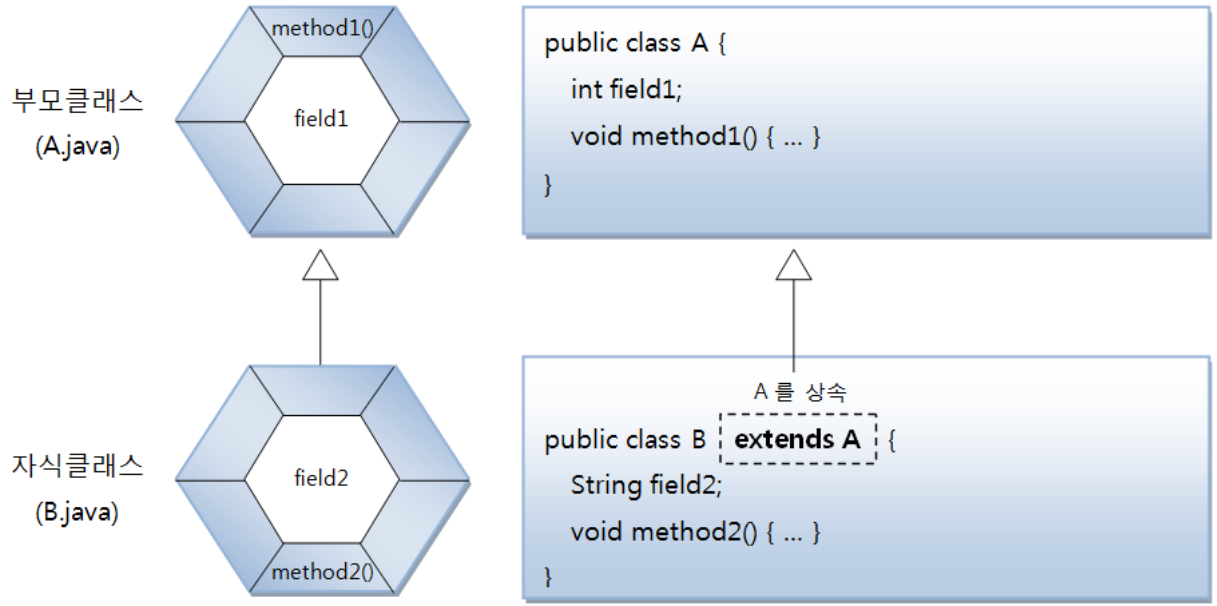
상속(Inheritance) 개념의 활용

- 상속의 효과
  - 부모 클래스를 재사용해 자식 클래스 개발 가능
  - 반복된 코드 중복 줄임
  - 유지 보수 편리성 제공
  - 객체의 다형성 구현 가능

- 상속 대상 제한
  - 부모 클래스의 private 접근 갖는 필드와 메소드 제외
  - 부모 클래스가 다른 패키지에 있을 경우, default 접근을 갖는 필드와 메소드도 제외

클래스 상속(extends)

- extends 키워드
  - 자식 클래스가 상속할 부모 클래스를 지정하는 키워드



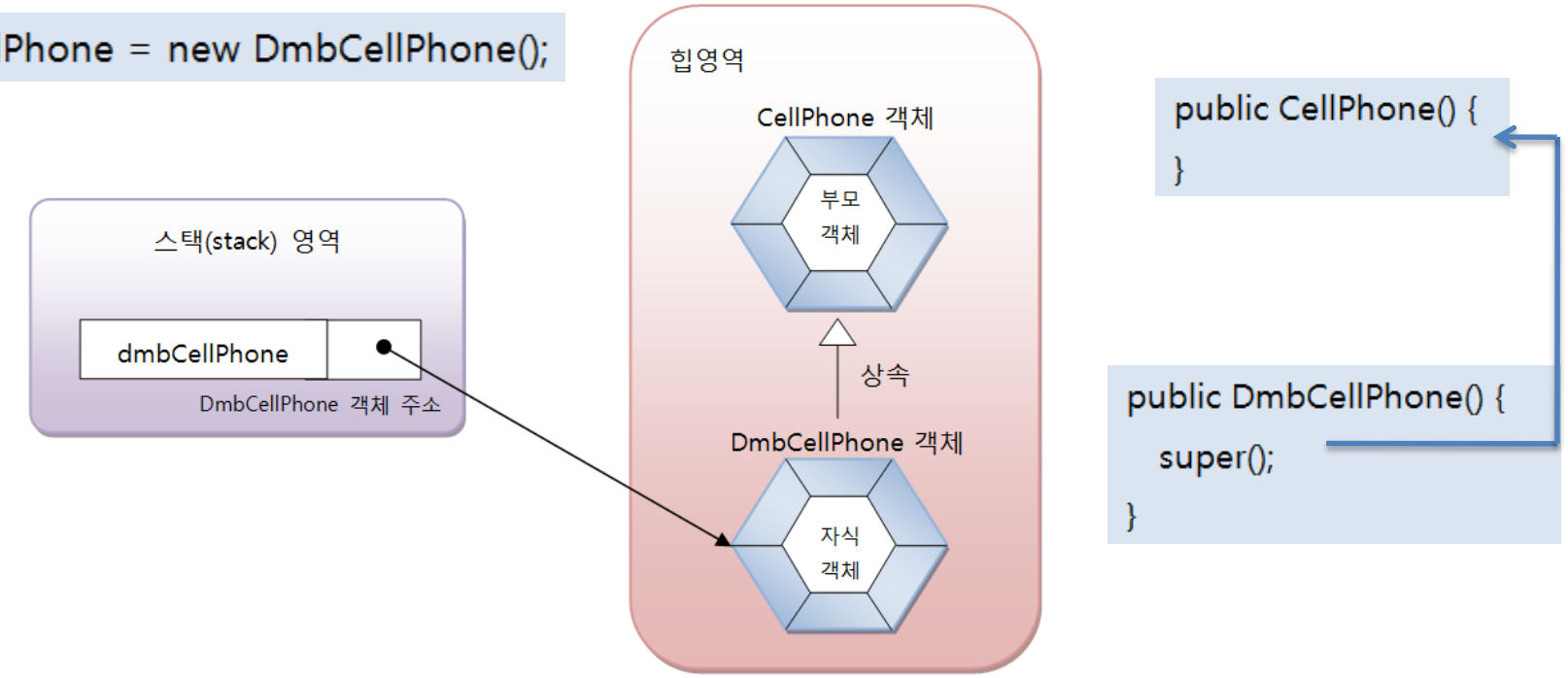
- 자바는 단일 상속 - 부모 클래스 나열 불가

```
class 자식클래스 extends 부모클래스1, 부모클래스2 {
}
```

부모 생성자 호출 super()

- 자식 객체를 생성하면 부모 객체도 생성되는가?
  - 부모 없는 자식 없음
  - 자식 객체 생성할 때는 부모 객체부터 생성 후 자식 객체가 생성 됨
  - 부모 생성자 호출 완료 후 자식 생성자 호출 완료
  - 부모 클래스에 기본(매개변수 없는) 생성자가 없다면 작성 필수

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




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

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

명시적인 부모 생성자 호출 super(...)

- 부모 객체 생성할 때, 부모 생성자를 선택해 호출 가능
  - 매개 값과 동일한 타입, 개수, 순서 맞는 부모 생성자 호출

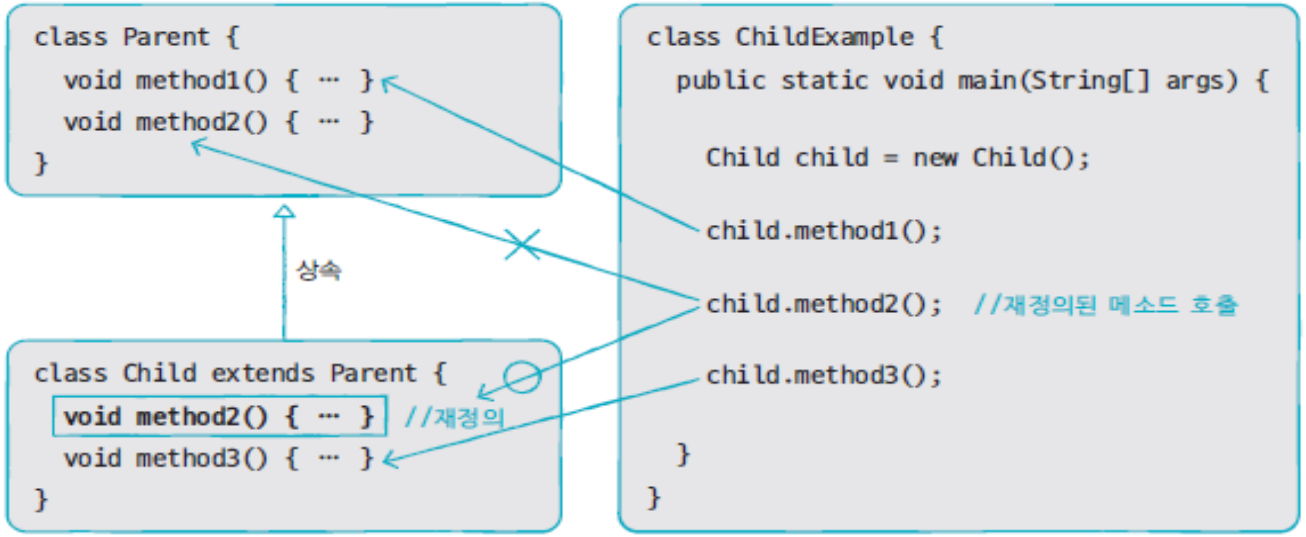
```
자식클래스( 매개변수선언, ... ) {
   super( 매개값, ... );
   ...
}
```

  - 부모 생성자가 없다면 컴파일 오류 발생

  - super(...)는 반드시 자식 생성자의 첫 줄에 위치해야 함.

메소드 재정의(오버라이딩(Overriding))

- 메소드 재정의(@Override)
  - 부모 클래스의 메소드가 자식 클래스에서 사용하기에 부적합할 경우 자식 클래스에서 수정하여 사용
    - 부모 메소드는 숨겨지는 효과 발생
  - 메소드가 재정의될 경우 부모 객체의 메소드가 숨겨지며,
  - 자식 객체에서 메소드 호출하면 재정의된 자식 메소드가 호출됨



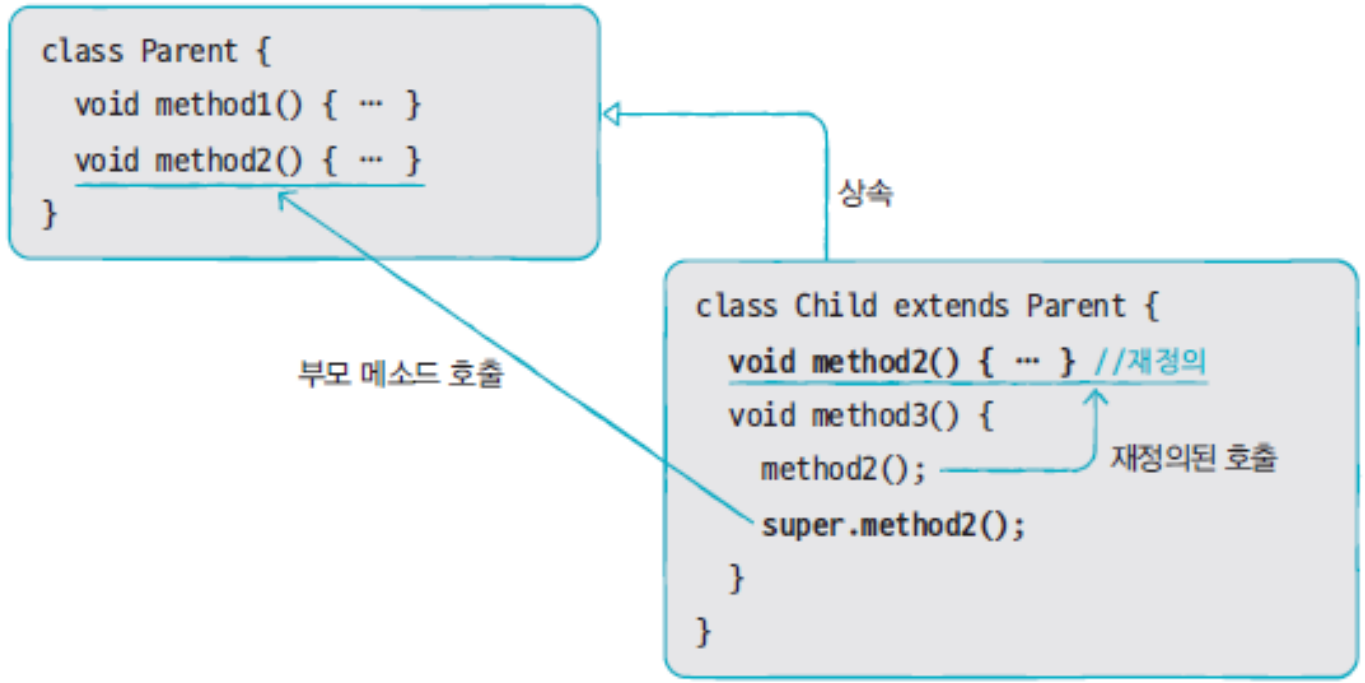
메소드 재정의 방법

- 메소드 재정의 조건
  - 부모 메소드와 동일한 시그니처를 가져야 함
  - 부모 클래스의 메소드보다 접근 제한을 더 좁은 범위로 변경할 수 없지만, 확장될 수는 있다.
  - 부모 클래스의 메소드보다 더 많은 예외를 던질 수 없다. (예외처리는 10장 참조)
  - 접근 제한자가 private 으로 정의된 메소드는 상속 자체가 안 된다.
  - static 메소드는 클래스에 속하는 메소드이기 때문에 상속이 안 된다.
  - final 메소드는 오버라이딩 할 수 없다.
- @Override 어노테이션
  - 컴파일러에게 부모 클래스의 메소드 선언부와 동일한지 검사 지시
  - 정확한 메소드 재정의를 위해 붙여주면 좋음

부모 메소드 호출

- 명시적으로 super 키워드 붙여 부모 메소드 호출
  - 자식 클래스 내부에서 재정의된 부모 클래스 메소드 호출해야 하는 경우
  - super는 부모 객체 참조(this는 자신 객체 참조)

```
super.부모메소드();
```



final 클래스

- final 키워드의 용도
  - 해당 선언이 최종 상태이며 수정될 수 없음을 의미

- 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 { … }
```

final 메소드, final 필드

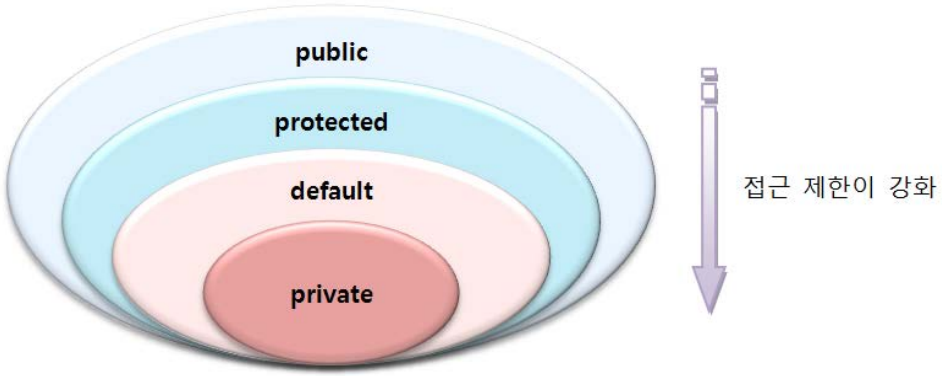
- 재정의할 수 없는 final 메소드
  - 부모 클래스에 선언된 final 메소드는 자식 클래스에서 재정의 할 수 없음

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

- final 필드
  - 수정이 불가능한 필드

protected 접근 제한자

- protected 접근 제한자
  - 상속과 관련된 접근 제한자
    - 같은 패키지: default와 동일
    - 다른 패키지: 자식 클래스만 접근 허용



접근 제한	적용할 내용	접근할 수 없는 클래스
public	클래스, 필드, 생성자, 메소드	없음
protected	필드, 생성자, 메소드	자식 클래스가 아닌 다른 패키지에 소속된 클래스
default	클래스, 필드, 생성자, 메소드	다른 패키지에 소속된 클래스
private	필드, 생성자, 메소드	모든 외부 클래스

다형성(多形性, Polymorphism)

- 같은 타입이지만 실행 결과가 다양한 객체를 대입해 이용할 수 있는 성질
- 부모 타입에는 자식 타입의 객체 대입 가능(자식 타입은 부모 타입으로 자동 타입 변환)

- 효과: 객체 부품화 가능

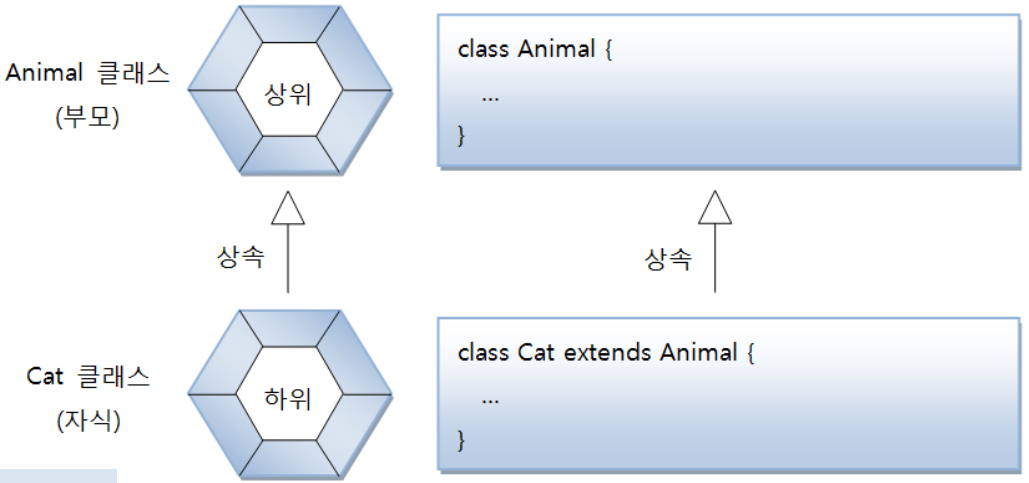
자동 타입 변환(Promotion)

❖ 자동 타입 변환(Promotion)

- 프로그램 실행 도중에 자동으로 타입 변환이 일어나는 것

자동 타입 변환

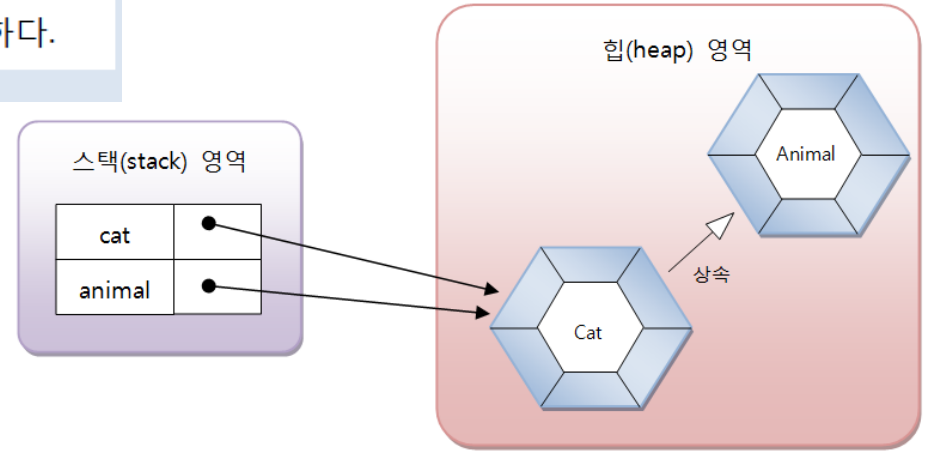
부모클래스 변수 = 자식클래스타입;




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

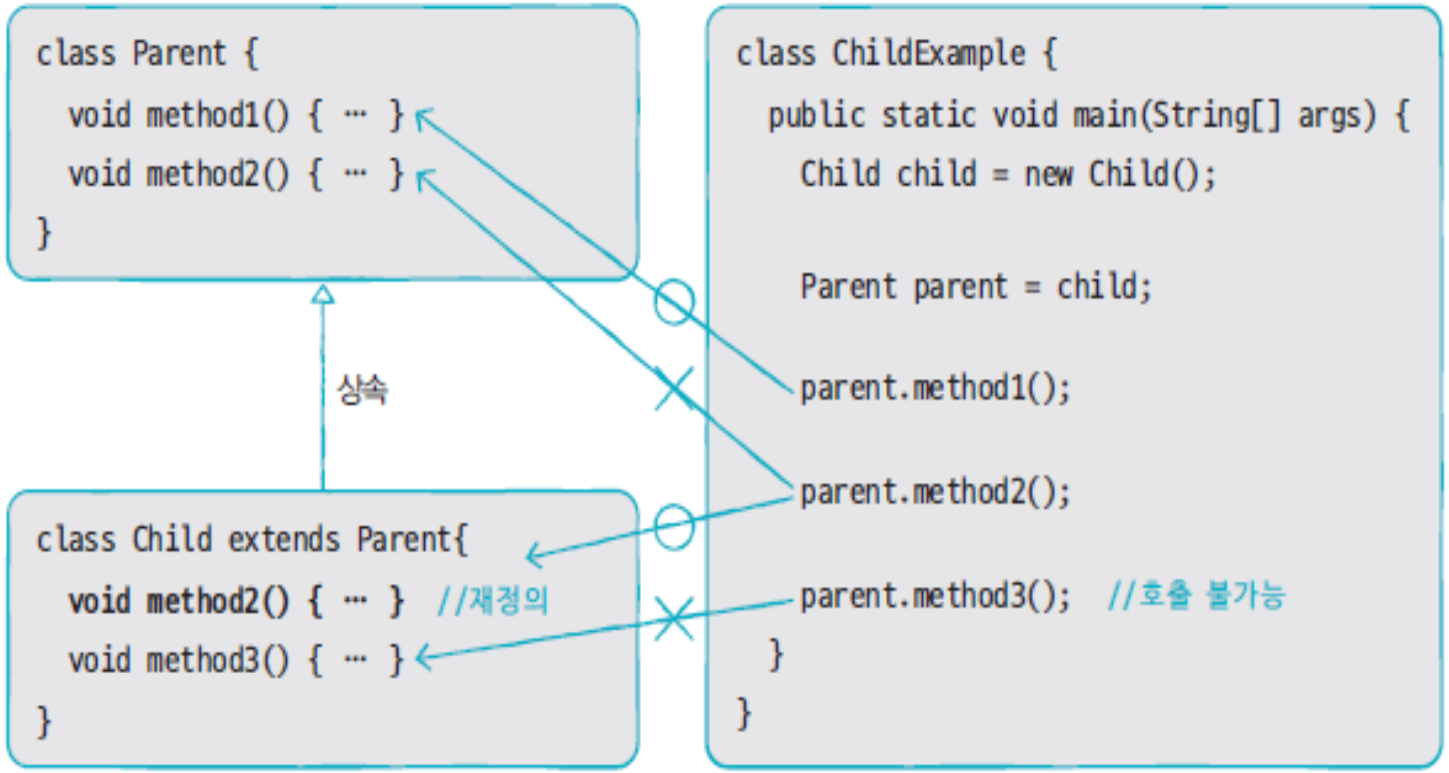
Animal animal = new Cat(); 도 가능하다.

```
cat == animal    //true
```



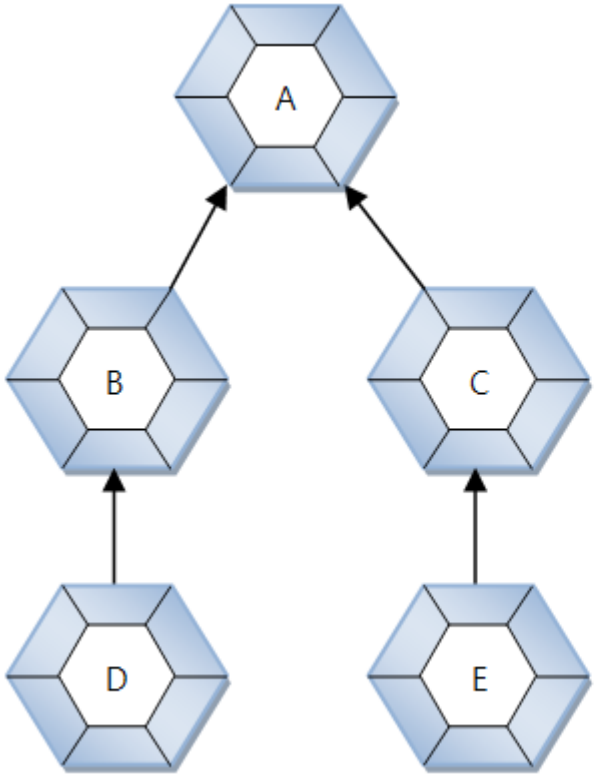
자동 타입 변환

- 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 됨.



상속관계에서 자동 타입 변환(Promotion)

- 자동 타입 변환(Promotion)
  - 바로 위의 부모가 아니더라도 상속 계층의 상위이면 자동 타입 변환 가능
    - 변환 후에는 부모 클래스 멤버만 접근 가능




```
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)
```

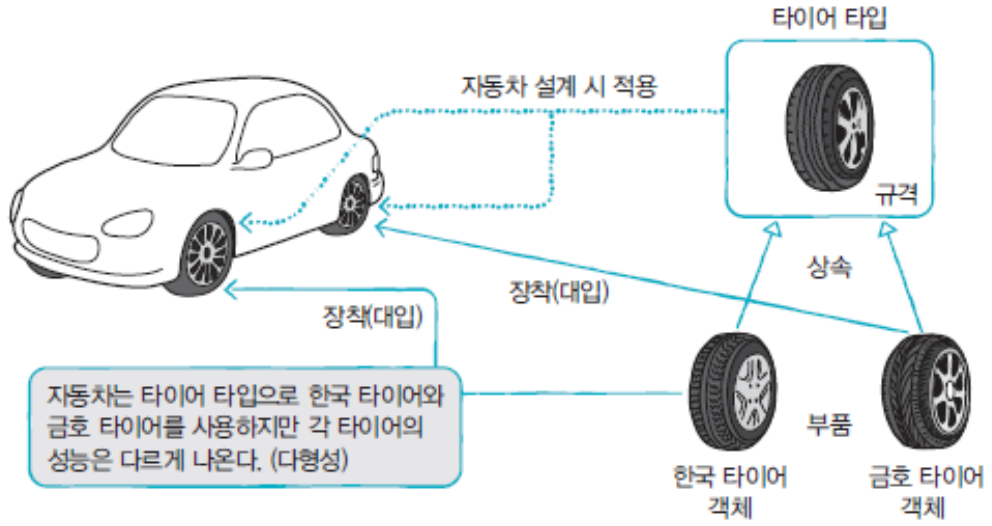
필드의 다형성

❖ 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();
```



다형성(Polymorphism) 객체 관리

❖ 하나의 배열로 객체 관리

```
class Car {
    Tire frontLeftTire = new Tire("앞왼쪽", 6);
    Tire frontRightTire = new Tire("앞오른쪽", 2);
    Tire backLeftTire = new Tire("뒤왼쪽", 3);
    Tire backRightTire = new Tire("뒤오른쪽", 4);
}
```

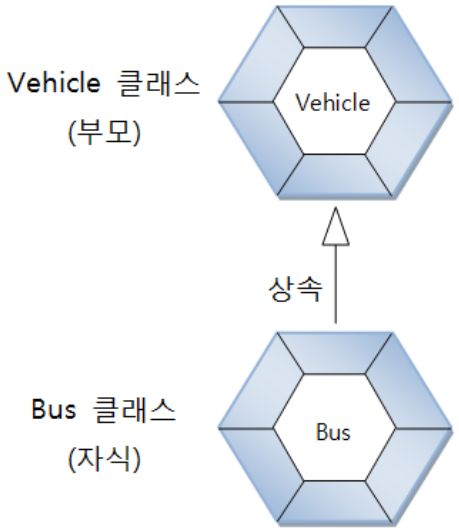
```
class Car {
    Tire[] tires = {
        new Tire("앞왼쪽", 6),
        new Tire("앞오른쪽", 2),
        new Tire("뒤왼쪽", 3),
        new Tire("뒤오른쪽", 4)
    };
}
```

```
tires[1] = new KumhoTire("앞오른쪽", 13);
```

```
int run() {
    System.out.println("[자동차가 달립니다.]");
    for(int i=0; i<tires.length; i++) {
        if(tires[i].roll()==false) {
            stop();
            return (i+1);
        }
    }
    return 0;
}
```

매개변수의 다형성

- 매개 변수를 부모 타입으로 선언하는 효과
  - 메소드 호출 시 매개 값으로 부모 객체 및 모든 자식 객체를 제공할 수 있음
  - 자동 타입 변환되어 자식에서 재 정의된 메소드가 호출 됨(매개변수의 다형성)




```
Driver driver = new Dirver();

Bus bus = new Bus();

driver.drive( bus );
```

자동 타입 변환 발생
Vehicle vehicle = bus;

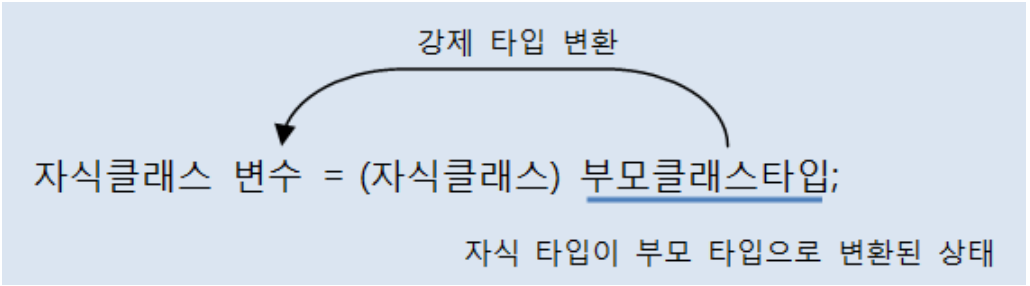
```
class Driver {

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

자식 객체

자식 객체가 재정의한 run() 메소드 실행

강제 타입 변환(Casting)

❖ 부모 타입을 자식 타입으로 변환하는 것

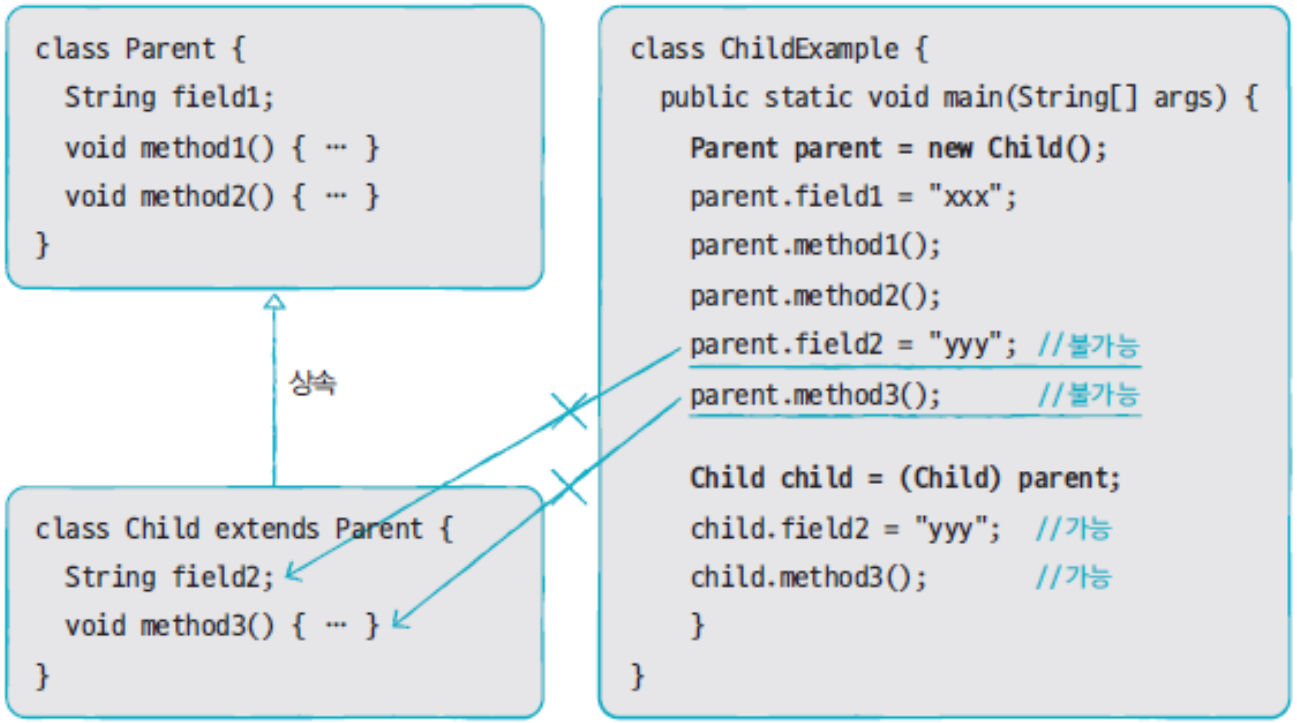


❖ 강제 타입 변환이 필요한 경우
- 자식 타입이 부모 타입으로 자동 변환되면 부모 타입에 선언된 필드와 메소드만 사용 가능
- 자식 타입에 선언된 필드와 메소드를 다시 사용해야 할 경우 강제 타입 변환

❖ 조건
- 자식 타입을 부모 타입으로 자동 변환 후, 다시 자식 타입으로 변환할 때 사용
- 타입 확인없이 강제 타입 변환 시도 시 ClassCastException 발생할 수 있음
- instanceof 연산자 통해 확인 후 안전하게 실행

강제 타입 변환(Casting) 예

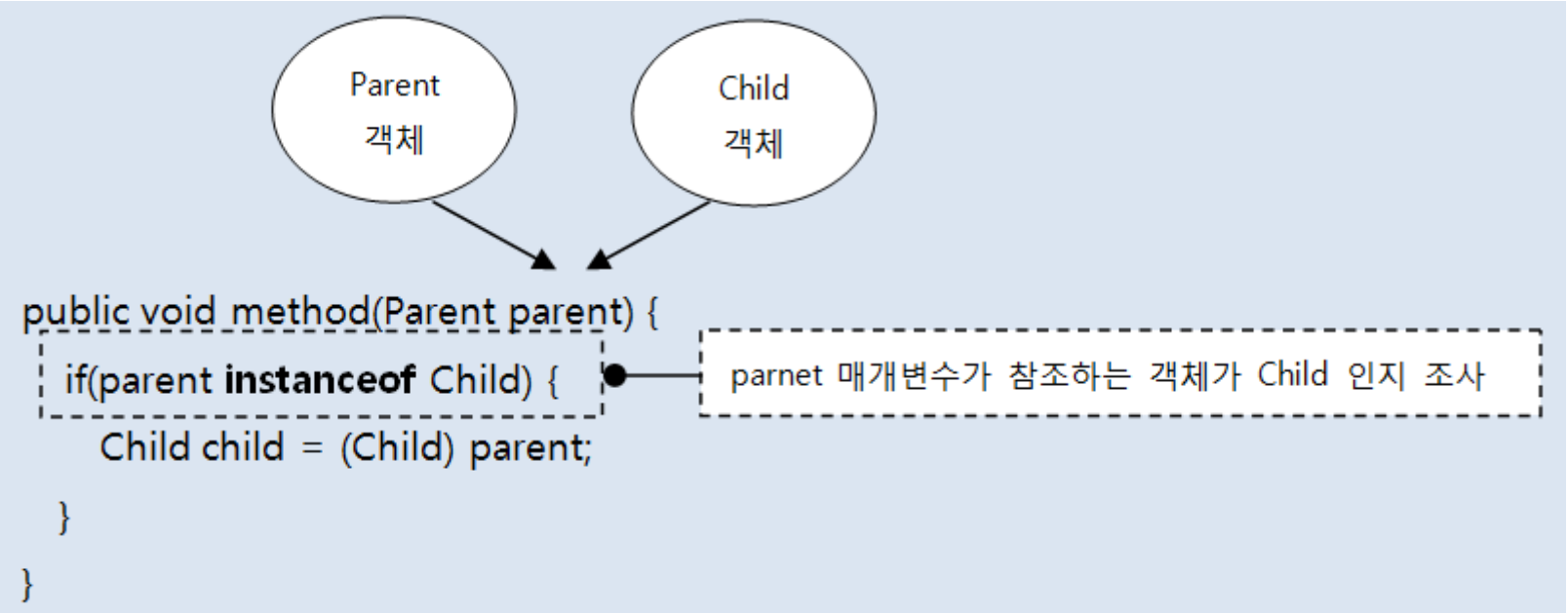
```
Parent parent = new Child();   //자동 타입 변환
Child child = (Child) parent;  //강제 타입 변환
```



instanceof 연산자

- 어떤 객체가 어느 클래스의 인스턴스인지 확인
- 메소드 내 강제 타입 변환이 필요한 경우 확인을 위해 사용

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

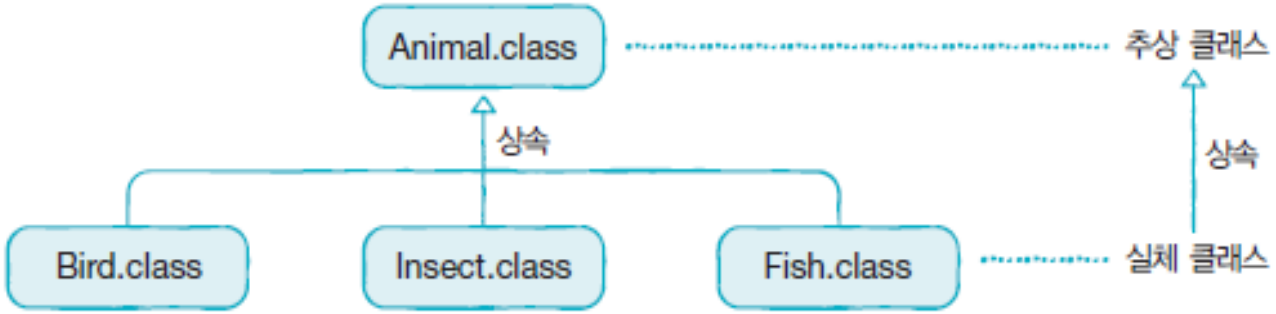


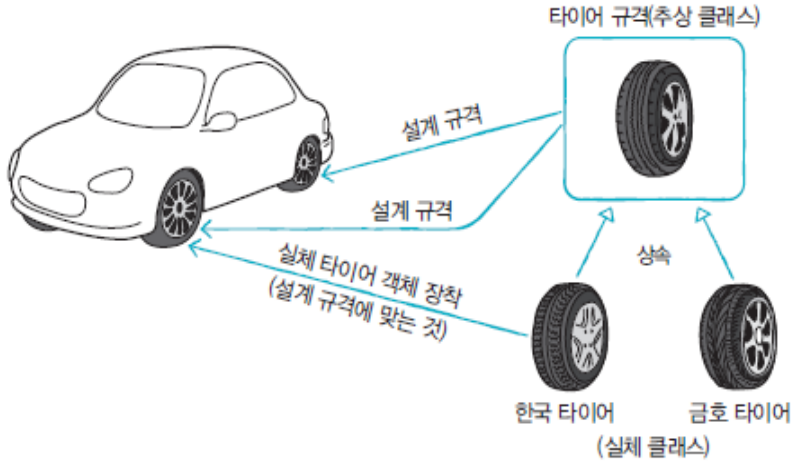
추상 클래스(abstract Class)

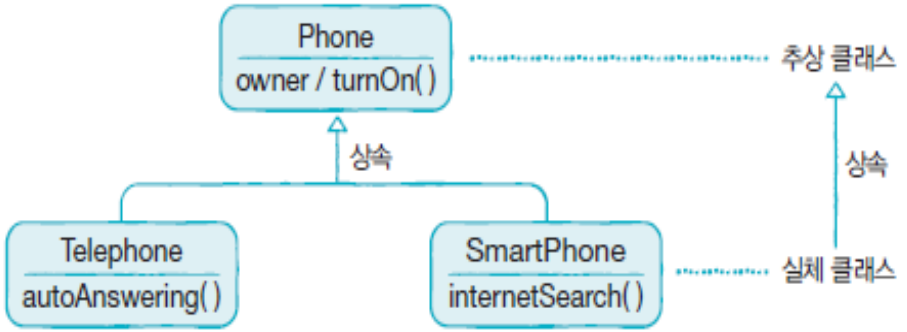
- abstract Class 개념
  - 여러 클래스의 공통된 특성(필드, 메소드)를 추출해서 선언한 것을 abstract Class라고 한다.
  - 예1: 새, 곤충, 물고기 -> 동물 (abstract Class)
  - 예2: 삼성, 현대, LG -> 회사 (abstract Class)
- abstract Class 용도
  - 실체 클래스 설계 규격을 만들고자 할 때
  - 실체 클래스가 반드시 가져야 할 공통된 필드와 메소드를 abstract Class에 미리 정의
  - abstract Class는 클래스의 설계 규격 : 객체 생성 불가능
- 실체 클래스는 abstract Class를 무조건 상속 받아 작성
  - 실체 클래스에서는 공통된 내용은 abstract Class를 통해 물려받고,
  - 다른 점만 추가적인 필드와 메소드로 선언

abstract Class와 실체 class

❖ abstract Class와 실체 클래스는 부모, 자식 클래스로서 상속 관계를 가짐







abstract Class 선언

❖ 추상 클래스 선언

- 클래스 선언에 abstract 키워드 사용

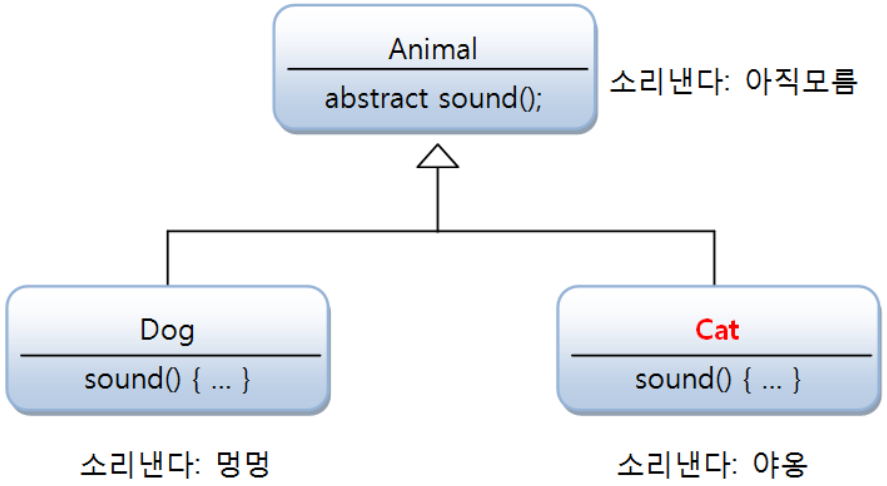
```
public abstract class 클래스 {
  //필드
  //생성자
  //메소드
}
```

- 직접 객체를 생성할 수 없지만 자식 객체가 생성될 때 객체화 됨.
  - New 연산자로 객체 생성하지 못하고 상속 통해 자식 클래스만 생성 가능
  - 자식 생성자에서 super(...) 형태로 추상 클래스의 생성자 호출

abstract 메소드와 오버라이딩(재정의)

- abstract 메소드
  - 메소드 선언만 통일하고 실행 내용은 실체 클래스마다 달라야 하는 경우
  - 예: 동물은 소리를 낸다. 하지만 실체 동물들의 소리는 제각기 다르다.

- 구현 방법
  - 추상 클래스에는 메소드의 선언부만 작성 (추상 메소드)
  - 실체 클래스에서 메소드의 실행 내용 작성(오버라이딩(Overriding))




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

```
public class Dog extends Animal {
    @Override
    public void sound() {
        System.out.println("멍멍");
    }
}
```

```
public class Cat extends Animal {
    @Override
    public void sound() {
        System.out.println("야옹");
    }
}
```

상속 중요 소스

- java_07장_상속_중요소스.pdf

적용 확인 학습 & 응용 프로그래밍

- 다음 파일에 있는 문제들의 해답을 스스로 작성 해 보신 후 개념 & 적용 확인 학습 영상을 학습 하시기 바랍니다.
  - java_07장_상속_ex.pdf

- 퀴즈와 과제가 출제되었습니다.
  - 응용 프로그래밍 영상을 학습하신 후 과제와 퀴즈를 수행 하시기 바랍니다.

학습 정리

- 상속, 메소드 재정의
- final 클래스, final 메소드
- 클래스 타입 변환, 자동 타입 변환, 강제 타입 변환
- instanceof 연산자
- 다형성(Polymorphism)
- abstract 클래스, abstract 메소드

Q & A

- "상속"에 대한 학습이 모두 끝났습니다.

- 모든 내용을 이해 하셨나요?
- 아직 이해가 안되는 내용이 있다면 다시 한번 복습하시기 바랍니다.
- 질문은 한림 SmartLEAD 쪽지 또는 e-mail 또는 전화상담을 이용하시기 바랍니다.

- 퀴즈와 과제가 출제되었습니다. 마감시간에 늦지 않도록 주의해 주세요.

- 다음 시간에는 "인터페이스"를 공부하도록 하겠습니다.

- 수고하셨습니다.^^