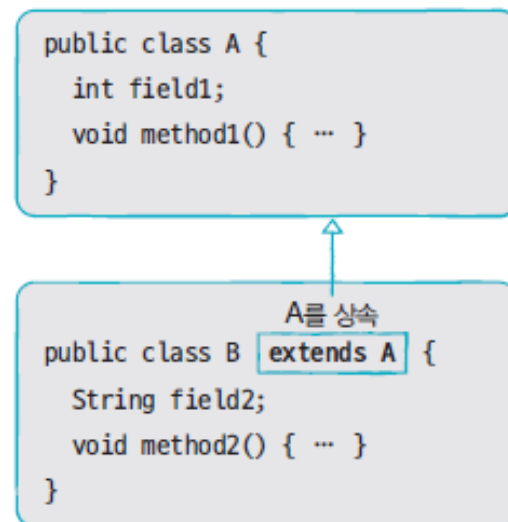
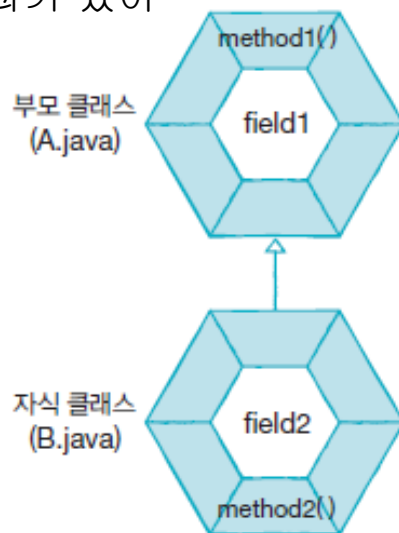


객체 지향 프로그램에서 부모 클래스의 멤버를 자식 클래스에게 물려줄 수 있다.

❖ 상속

- 이미 개발된 클래스를 재사용하여 새로운 클래스를 만들기에 중복되는 코드를 줄임
- 부모 클래스의 한번의 수정으로 모든 자식 클래스까지 수정되는 효과가 있어 유지보수 시간이 줄어듦



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();
```

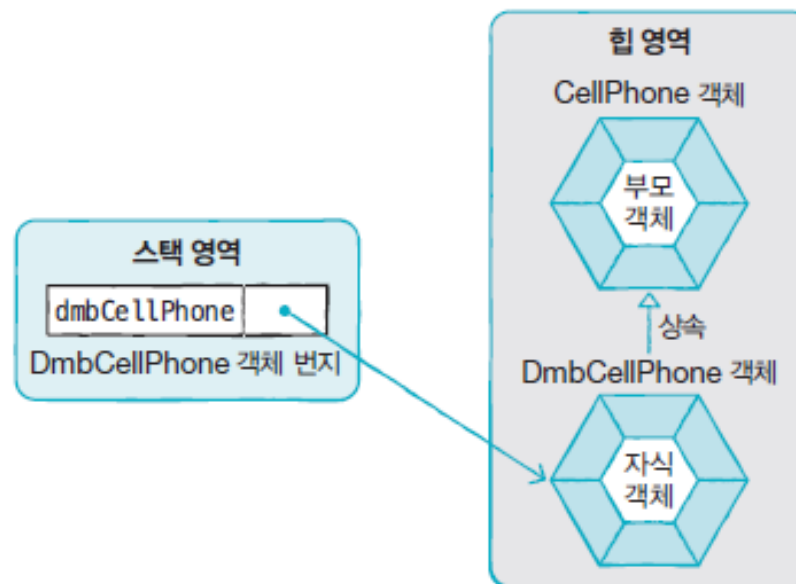
- 자식 생성자의 맨 첫 줄에서 부모 생성자가 호

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

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

- 명시적으로 부모 생성자 호출하려는 경우

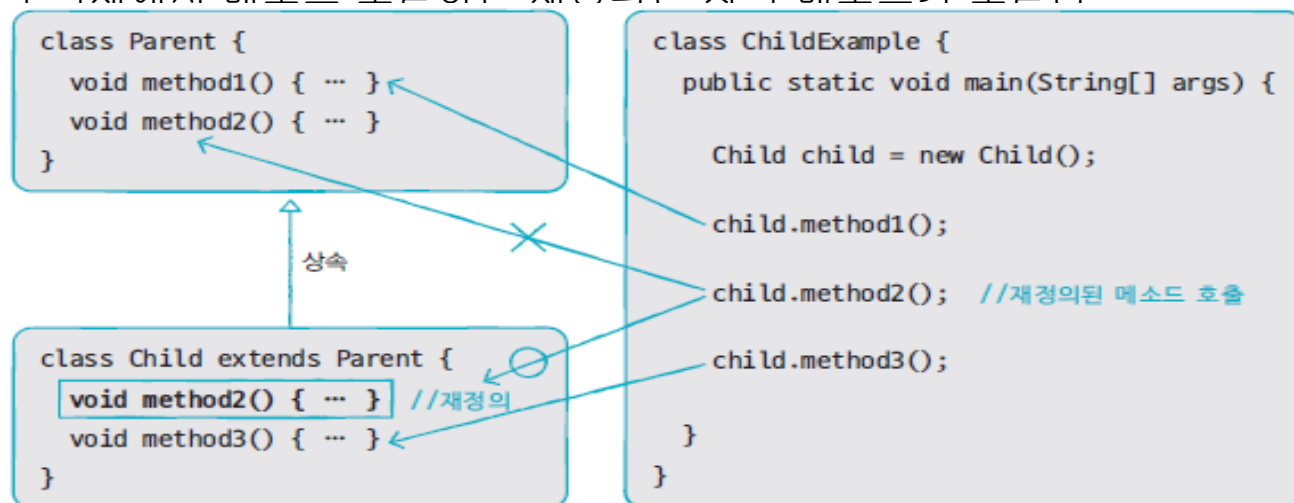
```
자식클래스( 매개변수선언, ... ) {  
    super( 매개값, ... );  
    ...  
}
```



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();
```

```
class Parent {  
    void method1() { ... }  
    void method2() { ... }  
}
```

상속

부모 메소드 호출

```
class Child extends Parent {  
    void method2() { ... } //재정의  
    void method3() {  
        method2();  
        super.method2();  
    }  
}
```

재정의된 호출

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, ...] ) { ... }
```

Quiz. Adult.java (class)

클래스 Person 의 필드

- int age;
- public String name;
- protected int height;
- private int weight;

클래스 Adult(실행)

- Person 클래스 상속
- private 인 weight 필드는 getter, setter 생성하여 접근 사용 가능
- 나이, 이름, 키, 몸무게를 호출하는 set()메서드 생성

출력예시)

나이:30

이름:홍길동

키:175

몸무게:99

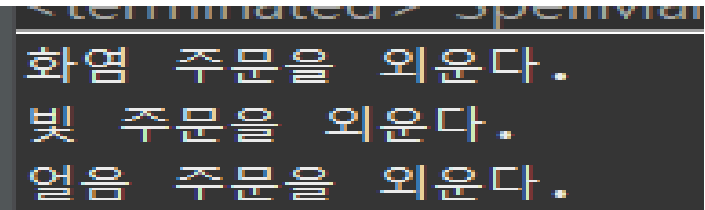
Quiz. SpellEx.java (class)

마법사가 3가지 주문을 외우는 코드를 작성하시오,

```
public class Spell {  
    public String casting() {  
        return "주문을 외운다.";  
    }  
}
```

Spell클래스를 상속하는 Fire(), Light(), Ice() 각 클래스와 메서드 생성

```
public class SpellEx {  
    public static void main(String[] args) {  
        Spell[] mage = new Spell[3];  
    }  
}
```

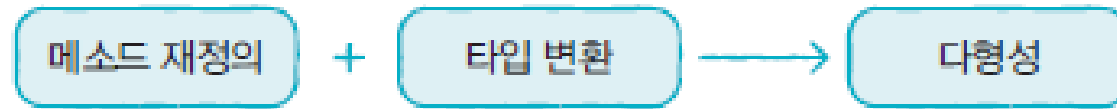


```
<terminated> SpellMain  
화염 주문을 외운다.  
빛 주문을 외운다.  
얼음 주문을 외운다.
```


기본 타입과 마찬가지로 클래스도 타입 변환이 있다.
이를 활용하면 객체 지향 프로그래밍의 다형성을 구현할 수 있다.

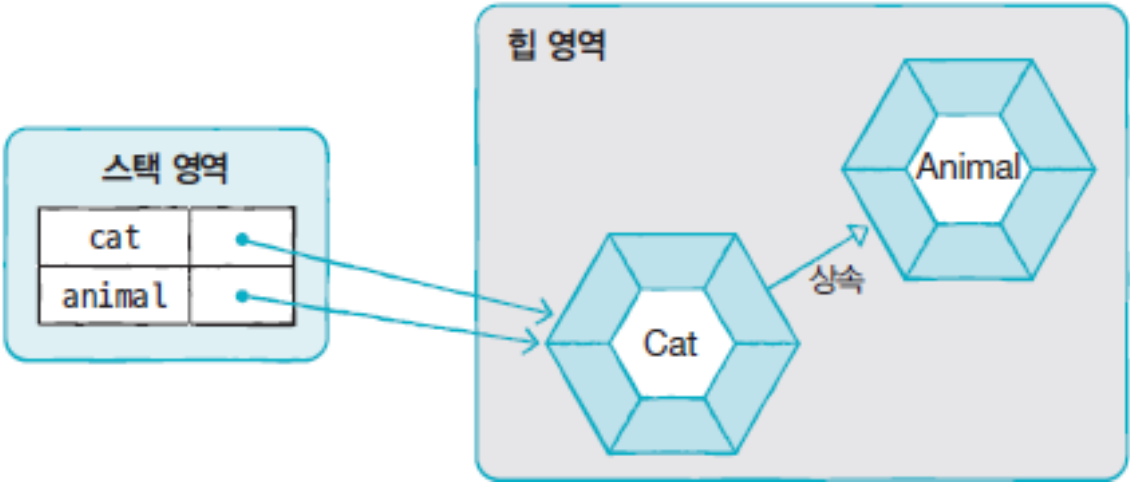
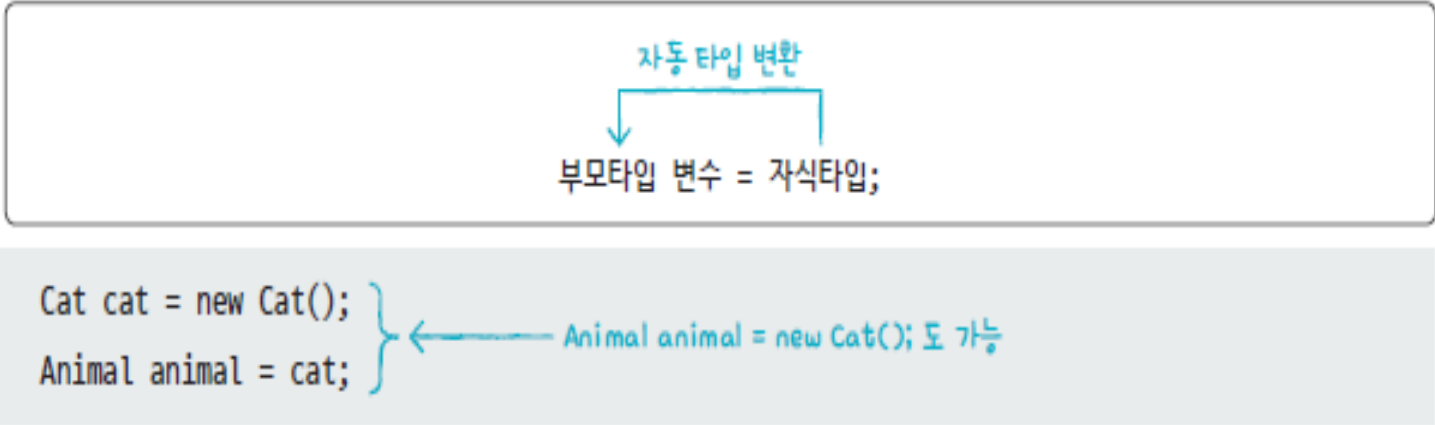
❖ 다형성

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



자동 타입 변환

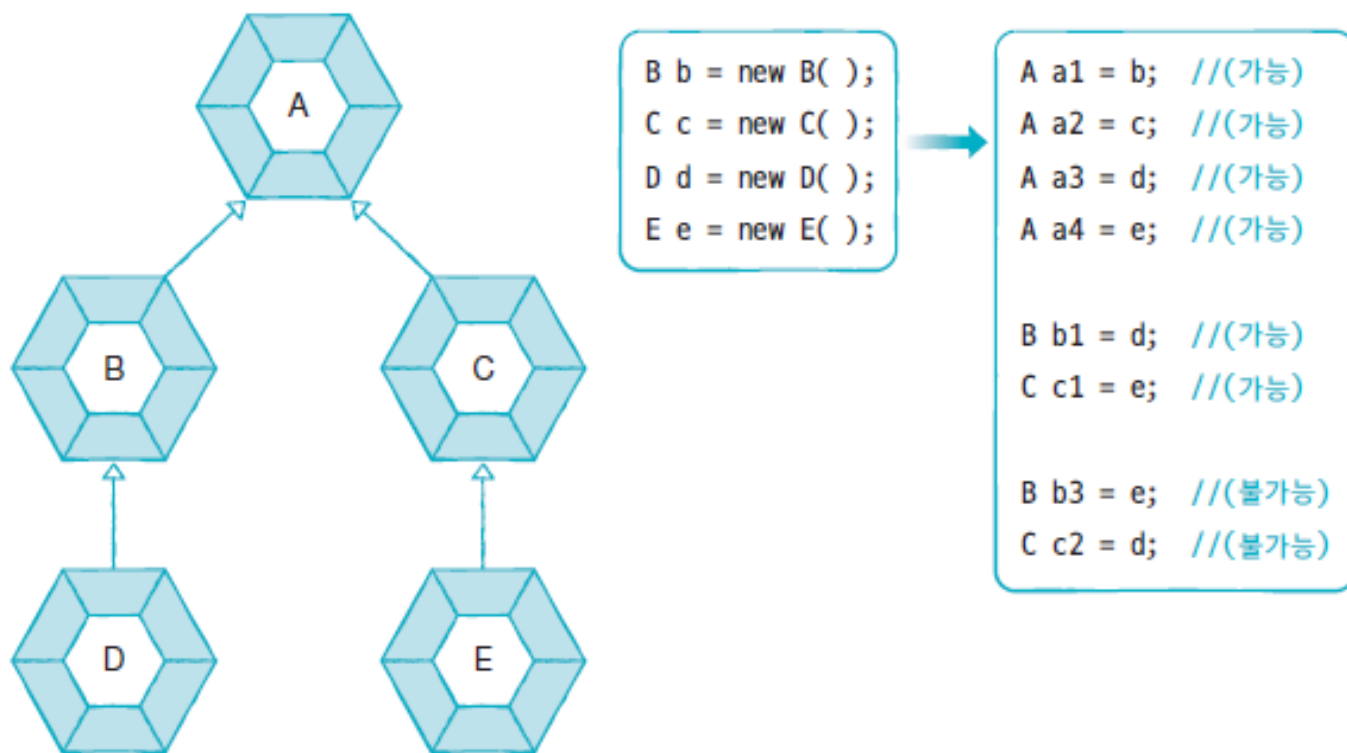
- ❖ 자동 타입 변환 (promotion)
 - 프로그램 실행 도중 자동으로 타입 변환 일어나는 것



cat == animal

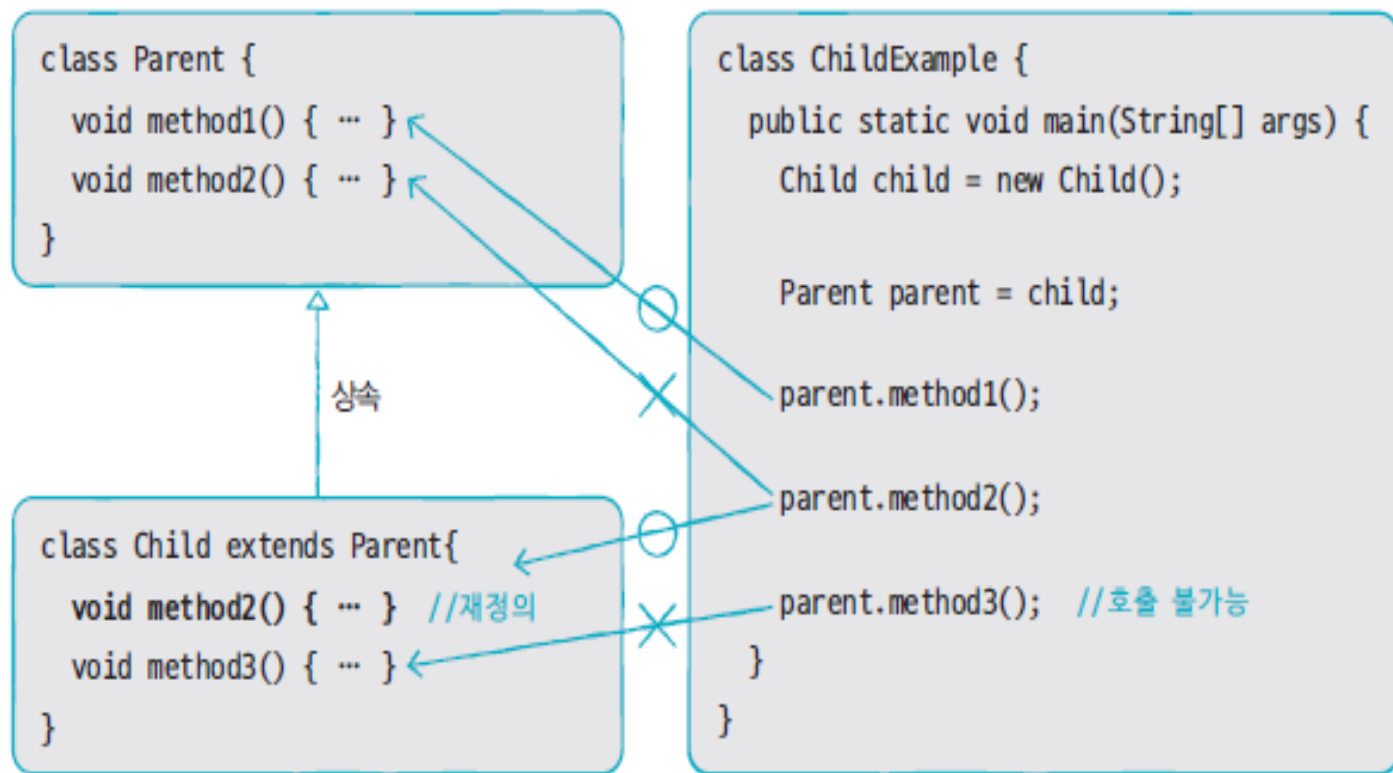
자동 타입 변환

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



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

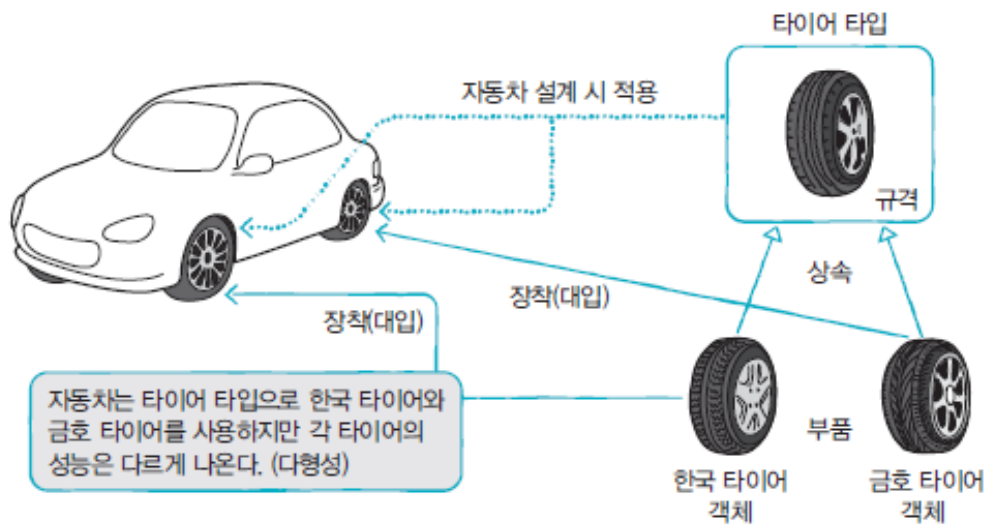


필드의 다형성

❖ 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();
```

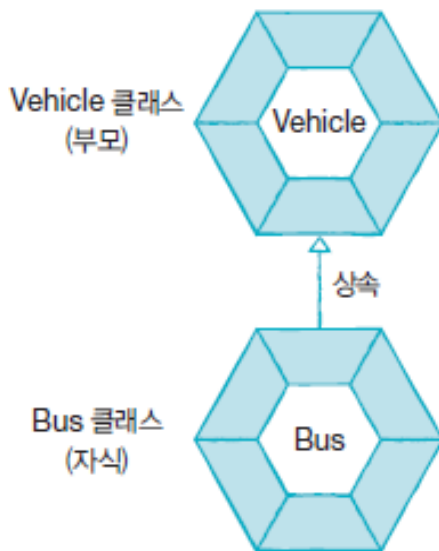
매개 변수의 다형성

❖ 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
```



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

Parent 매개 변수가 참조하는 객체가 Child인지 조사

Quiz. PersonEx.java (class)

클래스 Person 의 필드

- String phone;
- Phone에 대해 getter, setter 구성

클래스 Professor

- Person 클래스 상속
- 메서드 생성하여 아래와 같이 호출

출력예시)

Professor :010-111-1222 // Professor의 객체

Professor :010-111-1222 // Person의 객체를 타입변환하여 호출

* 일정을 출력하는 스케줄러를 작성해봅시다.

부모클래스 (Event.java)

```
// OneDay/Duration/Deadlined Event의 공통 멤버를
// 관리하는 상위 클래스
public String title;
public Event(String title) {
    this.title = title;
}
```

메인 클래스 (scheduler.java)

```
public static void main(String[] args) {
    Event[] evt = new Event[3];
    myDate date1 = new myDate(2021,6,17);
    myDate date2 = new myDate(2021,6,04);
    myDate date3 = new myDate(2021,8,30);
    evt[0] = new OneDay("오늘",date1);
    evt[1] = new Duration("java",date2,date3);
    evt[2] = new Deadlined("마감날짜",date3);
    for(int i=0; i<evt.length;i++) {
        System.out.println( "일 정: "+ evt[i].toString());
    }
}
```

```
일 정: 오늘, 2021/6/17
일 정: java, 시작 날짜: 2021/6/4, 마감 날짜: 2021/8/30
일 정: 마감날짜, ~2021/8/30
```

자식 클래스

OneDay 클래스

필드 : title(상속), myDate date
생성자 : 매개변수 2개
메서드 : toString

Duration 클래스

필드 : title(상속), myDate begin, myDate end
생성자 : 매개변수 3개
메서드 : toString

Deadlined 클래스

필드 : title(상속), myDate deadline
생성자 : 매개변수 2개
메서드 : toString

myDate 클래스

필드 : int year, month, day
생성자 : 매개변수 3개
메서드 : toString

Quiz. 베이커리 카페에서 커피와 빵을 사먹는 제임스 (CoffeShop패키지)

- 커피숍에 아메리카노와 카페라테, 카푸치노가 있습니다. (가격은 부모 클래스로 생성)
- 빠리바게트에 크림도넛, 단팥빵, 모카빵이 있습니다(가격은 부모 클래스로 생성)
- 제임스가 커피숍에서 음료수를 구매했습니다. 구매 총액을 출력합니다.
- 제임스가 빠리바게트에서 빵을 구매했습니다. 구매 총액을 출력합니다.

Quiz. 베이커리 카페에서 커피와 빵을 사먹는 제임스 (CoffeShop패키지)

부모클래스 (Product.java)

```
// americano, caffelatte, capucchino, CreamDonut,
RedbeanBread, MoccaBread  공통 멤버를 관리하는 상위클래스
public class Product {
    int price;
    Product(int price) {
        this.price = price;
    }
}
```

Paribaguette 클래스

필드 : Product cd, rb, mb

생성자 : 매개변수 3개,

CreamDonut, RedbeanBread, MoccaBread인스턴스 생성

CoffeeShop 클래스

필드 : Product a, c, ca

생성자 : 매개변수 3개,

Americano, Caffelatte, Capucchino 클래스의 인스턴스 생성

James 클래스

필드 : int total

생성자 : 기본

메서드 : buy() - 구입합계

자식클래스

: 부모 클래스에서 상속받은 금액만 입력 받는 클래스

Americano

Caffelatte

Capucchino

CreamDonut

RedbeanBread

MoccaBread

생성자 : 매개변수 1개

실행 클래스 (buy.java)

단팥빵의 가격:1500

크림도넛의 가격:1000

모카빵의 가격:2000

카푸치노의 가격:3000

아메리카노의 가격:1000

카페라떼의 가격:2000

제임스가 먹은 빵의 총 가격은? 4500

제임스가 마신 커피의 총 가격은? 6000

Quiz. 베이커리 카페에서 커피와 빵을 사먹는 제임스 (CoffeShop패키지)

부모클래스 (Product.java)

```
// americano, caffelatte, capucchino, CreamDonut,
RedbeanBread, MoccaBread  공통 멤버를 관리하는 상위클래스
public class Product {
    int price;
    Product(int price) {
        this.price = price;
    }
}
```

Paribaguette 클래스

필드 : Product cd, rb, mb

생성자 : 매개변수 3개, 하위 클래스의 인스턴스 생성

하위 클래스 : CreamDonut, RedbeanBread, MoccaBread

CoffeeShop 클래스

필드 : Product a, c, ca

생성자 : 매개변수 3개, 하위 클래스의 인스턴스 생성

하위 클래스 : Americano, Caffelatte, Capucchino

James 클래스

필드 : int total

생성자 : 기본

메서드 : buy() - 구입합계

자식클래스

: 부모 클래스에서 상속받은 금액만 입력 받는 클래스

Americano

Caffelatte

Capucchino

CreamDonut

RedbeanBread

MoccaBread

생성자 : 매개변수 1개

실행 클래스 (buy.java)

단팥빵의 가격:1500

크림도넛의 가격:1000

모카빵의 가격:2000

카푸치노의 가격:3000

아메리카노의 가격:1000

카페라떼의 가격:2000

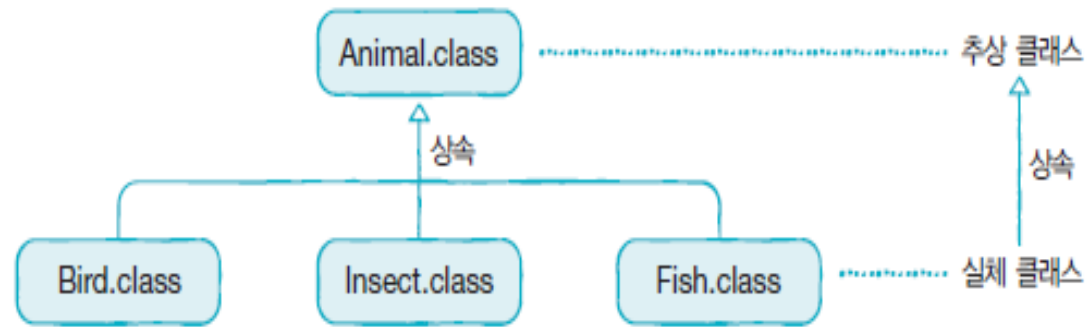
제임스가 먹은 빵의 총 가격은? 4500

제임스가 마신 커피의 총 가격은? 6000

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

❖ 추상 클래스

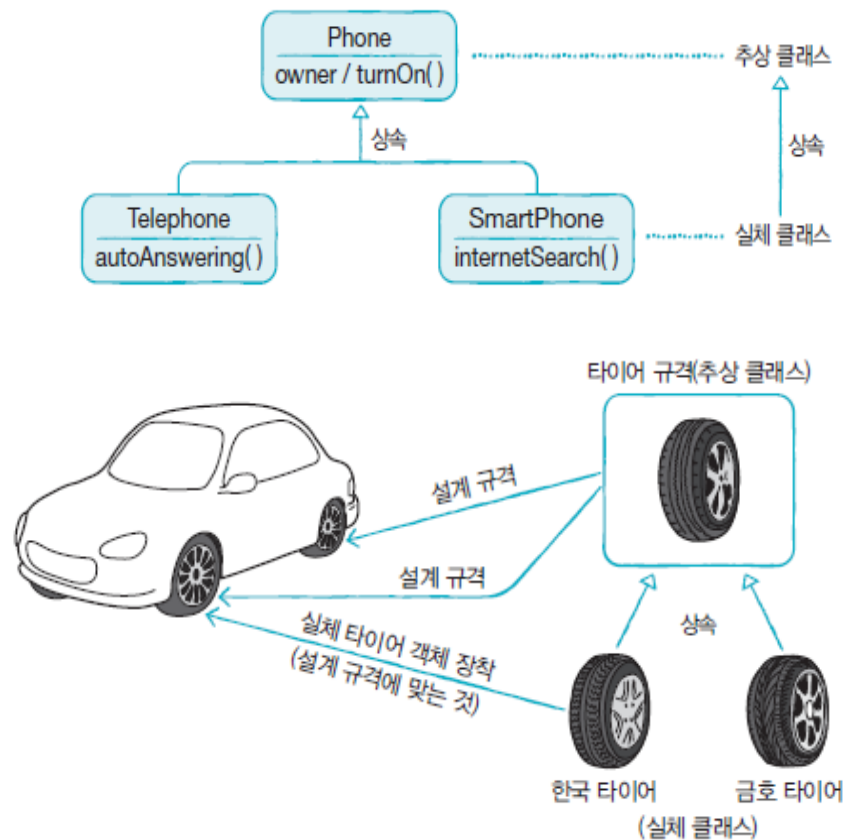
- 실체 클래스(객체 생성용 클래스)들의 공통적인 특성(필드, 메소드)을 추출하여 선언한 것
- 추상 클래스와 실체 클래스는 부모, 자식 클래스로서 상속 관계를 가짐



추상 클래스의 용도

❖ 추상 클래스의 용도

- 실체 클래스에 반드시 존재해야 할 필드와 메소드의 선언(실체 클래스의 설계 규격 - 객체 생성용이 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 만듦(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

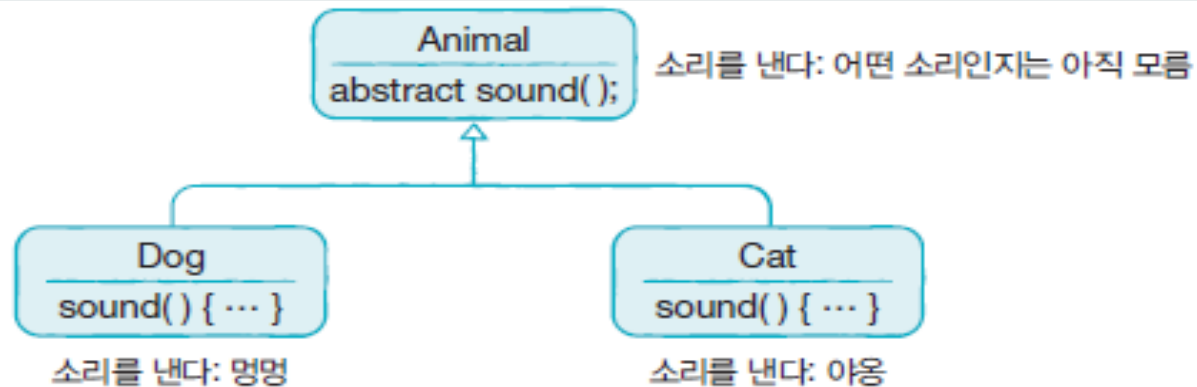
추상 메소드와 재정의

❖ 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();  
}
```



Quiz. CalculatorEx.java

```
public abstract class Calculator {  
    public abstract int add(int a, int b);  
    public abstract int subtract(int a, int b);  
    public abstract double average(int[] a);  
}
```

위의 Calculator클래스를 상속받아 CalcMethod클래스에서 메소드들을 재정의 하고 CalculatorEx클래스를 통해 결과를 출력하세요.

출력예시) `ex> a=2, b=3, int[] array = { 1, 2, 3, 4, 5 };`

add: 5

subtract: -1

average: 3.0

Quiz. HttpServletExample.java

```
public abstract class HttpServlet {  
    public abstract void service();  
}
```

위의 HttpServlet 클래스를 상속받아 LoginServlet 클래스와 FileDownloadServlet 클래스에서 메소드들을 재정의 하고 HttpServletExample 클래스를 통해 결과를 출력하세요.

출력예시)

로그인 합니다.
파일 다운로드 합니다.