객체지향 프로그램 정리

프로그램에서 **"변하지 않는 것은 '변화' 뿐"**이라고 프로그램은 어떤 이유에서 꼭 변경되고 확장되고 수정되며 재사용됩니다. **프로그램은 확장성과 재사용성 유지보수성 등이 아주 중요합니다.**

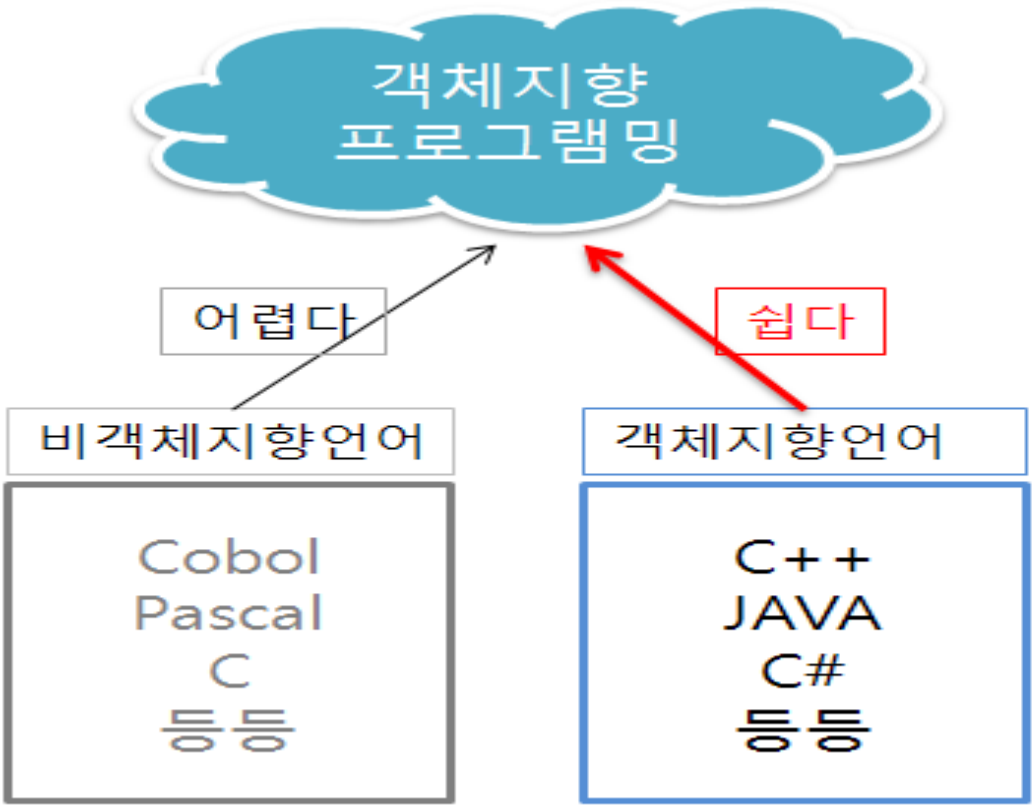
객체지향은 확장성과 재사용성이 뛰어납니다. 또 여러 가지 좋은 특성이 있습니다.

그래서 현재 프로그램 세계는 **객체지향이 기본이고 필수입니다.대세는 객체지향이다!!**

Java 언어도 객체지향을 지원하는 객체지향언어입니다.

객체지향은 사고이며 방법론이므로 객체지향언어가 아니라고 해서 객체지향 프로그래밍을 하지 못한다는 말이 아닙니다.

다만, 객체지향언어로 객체지향 프로그래밍을 더 쉽게 더 잘할 수 있다는 이야기입니다.



비 객체지향언어로 객체지향 프로그램을 만들 수 있지만 99.9% 불가능합니다. 그래서 객체지향 프로그램을 쉽게 만들 수 있는 객체지향 언어로 프로그래밍을 하는 것입니다.

Java 언어가 객체지향 언어지만 객체지향을 이해하지 못하고 객체지향의 장점을 이용하지 못한다면 비 객체지향 언어보다 나쁜 프로그래밍이 됩니다. 명심해야 합니다.

쉽게 말하면 '무늬만 객체지향 프로그램'을 만들 수 있다는 것입니다.

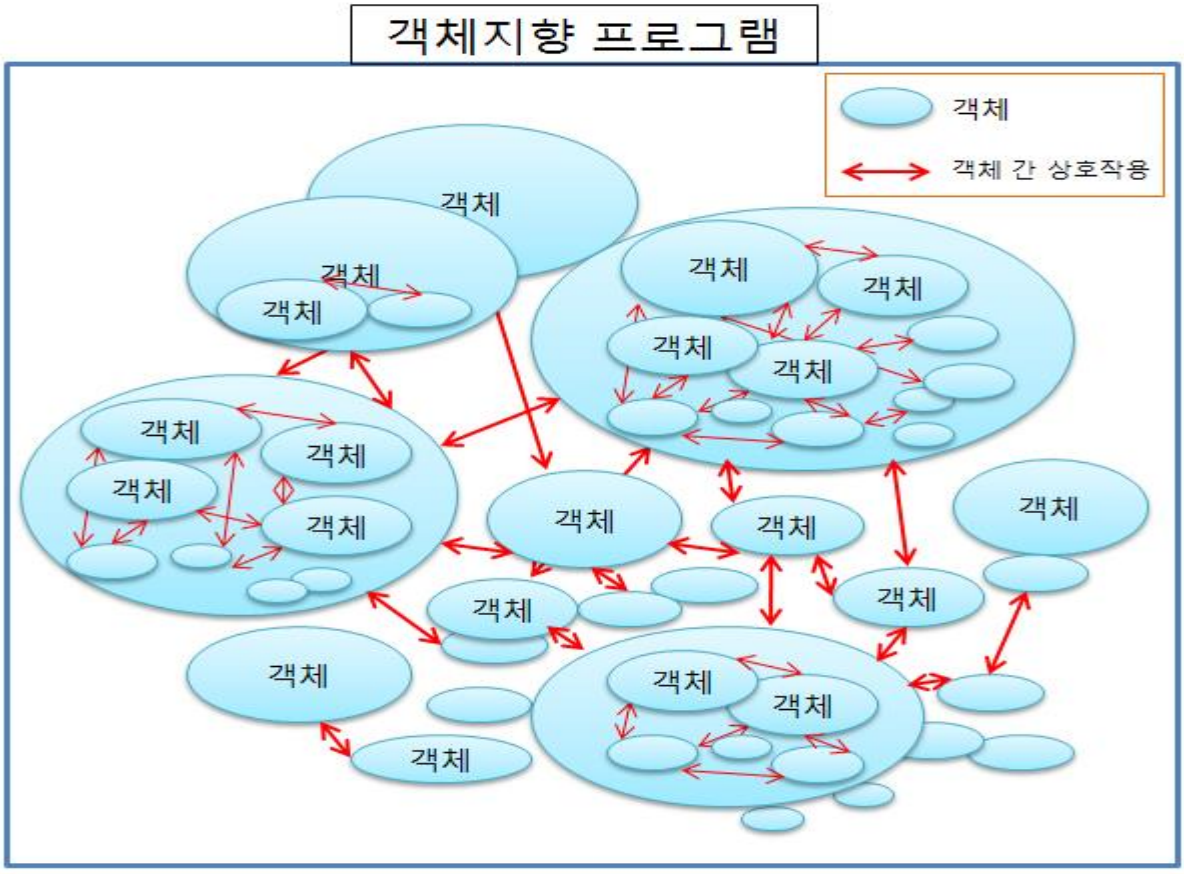
처음 객체지향언어를 공부하는 초보자들이 자주 하는 실수입니다. 객체지향은 급하게 생각하지 말고 꾸준히 꾸~준히 공부해야 합니다.

이제부터 객체지향 세계의 맛을 보도록 하겠습니다. 객체지향은 참 **멋진 놈**입니다.

우리는 이번 시간에 **'객체'(object), '클래스'(class), '인스턴스'(instance), '추상화'(abstraction), '상속'(inheritance), '캡슐화'(encapsulation), '다형성'(polymorphism), '메시지'(message), '인터페이스'(interface)**를 배우게 됩니다.

1, 객체(object)

객체지향 프로그래밍은 객체 간의 상호작용으로 만들어진 프로그램입니다.
그래서 객체지향에서 객체는 아주 중요합니다.



객체란 무엇일까요?
'객체'라는 단어를 우리가 잘 사용하지 않을 뿐 우리는 객체들 속에서 객체들과 생활하고 있습니다.
객체는 우리가 아는 거의 모든 것입니다.
예로 컴퓨터, 책상, 책, 의자, 나무, 숲, 시계, 전등, 사람 등등...

우리가 보고 만지고 느끼는 것 외에도 객체가 될 수 있습니다.
예로 비행경로, 비행시간, 날씨 정보, 인간관계, 생각, 관념 등등...

그래서 객체는 크게 둘로 분류할 수 있습니다.

- **물리적 객체 : 현실 세계에 보고 만질 수 있는**
- **개념적 객체 : 개념적인**

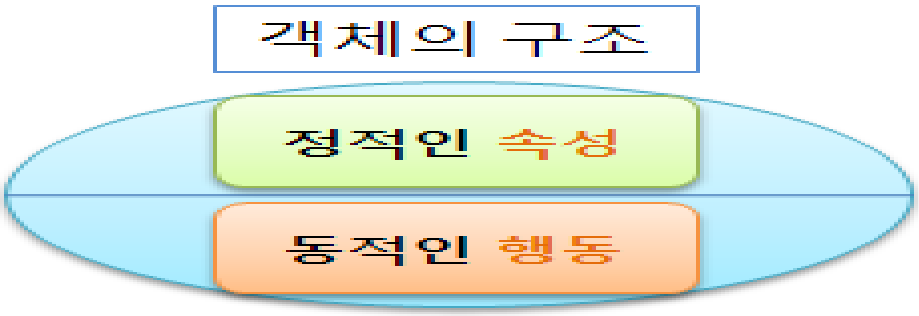
프로그램에서 객체를 굳이 분류할 필요는 없습니다. 필요에 따라 사용하거나 사용하지 않습니다.

1.1, 객체의 구조

객체(object)는 두 가지의 구성요소를 갖습니다.

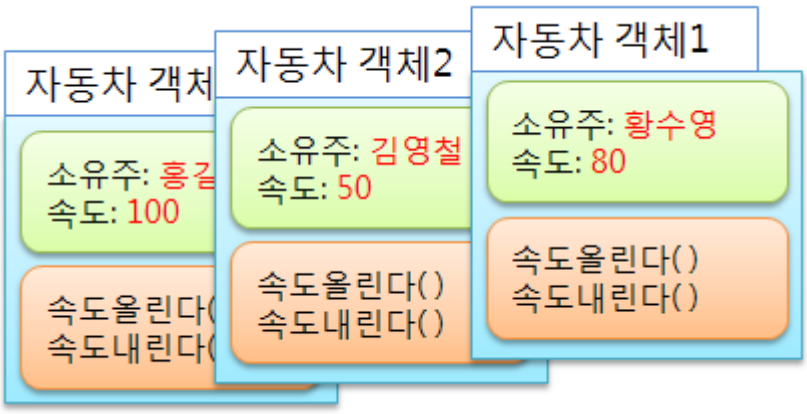
- **정적인 요소 : 상태, 속성, 필드, 아는 것, 변수**
- **동적인 요소 : 행위, 연산, 메소드, 하는 것, 함수**

책마다 약간씩 다르게 바라보기도 하지만 우리는 모두 같게 취급합니다.
<사실, 상태, 속성, 특성이란 용어가 영문으로 property, attribute 라는 용어를 번역한 것입니다. 그래서 프로그래밍 언어마다 약간씩 중복되고 구분되어 사용되므로 조금 어지럽습니다. >



객체의 구조로 보면
객체는 속성들과 행동들의 집합입니다.

'자동차'라는 객체를 예로 볼까요?
자동차는 **상태**로 소유주와 속도를 가질 수 있으며 **행동**으로 속도를 올리고 내릴 수 있습니다.
그림으로 보면 아래와 같습니다.



꼭 알아야 하는 객체의 **중요한 특징**이 있습니다.

객체는 **독립적으로 존재**하며 **다른 객체와 구분**된다는 것입니다.

그림처럼 자동차 객체 1 과 자동차 객체 2 가 독립적으로 존재합니다.

자동차 객체 1 의 속도를 높이면 자동차 객체 1 의 속도만 올라가고 자동차 객체 2 의 속도나 다른 객체의 속도는 변하지 않습니다.

당연한 이야기지만 객체에서 아주 중요한 이야기입니다.

그림에서 보면 자동차는 객체일까요? 정답은 아닙니다. 다시 말하지만, 객체는 다른 객체와 구분되어 독립적으로 존재해야 합니다. 자동차객체 1, 자동차객체 2, … 자동차객체 n, 하나하나는 객체지만 위 그림의 자동차는 자동차 하나하나를 구분하지 않으므로 객체라 볼 수 없습니다. 자동차는 자동차객체들의 공통적인 개념과 정의, 틀을 표현하므로 클래스에 해당합니다. 클래스는 다시 공부하겠습니다.

객체는 독립적으로 존재하며 다른 객체와 구분됩니다.(각각의 상태와 행동을 가집니다.)

- 자동차 객체 1
  - 상태 : 소유주 - 황수영, 속도 - 80
  - 행동 : 자동차 객체 1 의 차를 속도올린다. 자동차 객체 1 의 차를 속도내린다.
- 자동차 객체 2
  - 상태 : 소유주 - 김영철, 속도 - 50
  - 행동 : 자동차 객체 2 의 차를 속도올린다. 자동차 객체 2 의 차를 속도내린다.
- 자동차 객체 k .....
- 자동차 객체 n
  - 상태 : 소유주 - … , 속도 - …
  - 행동 : 자동차 객체 n 의 차를 속도올린다. 자동차 객체 n 의 차를 속도내린다.

여기서 자동차 객체 1, 자동차 객체 2, …, 자동차 객체 n 을 자동차의 **인스턴스(instance)**라 합니다.

인스턴스(자동차 객체)란 클래스(자동차)의 실체(생성된 진짜 객체)입니다.

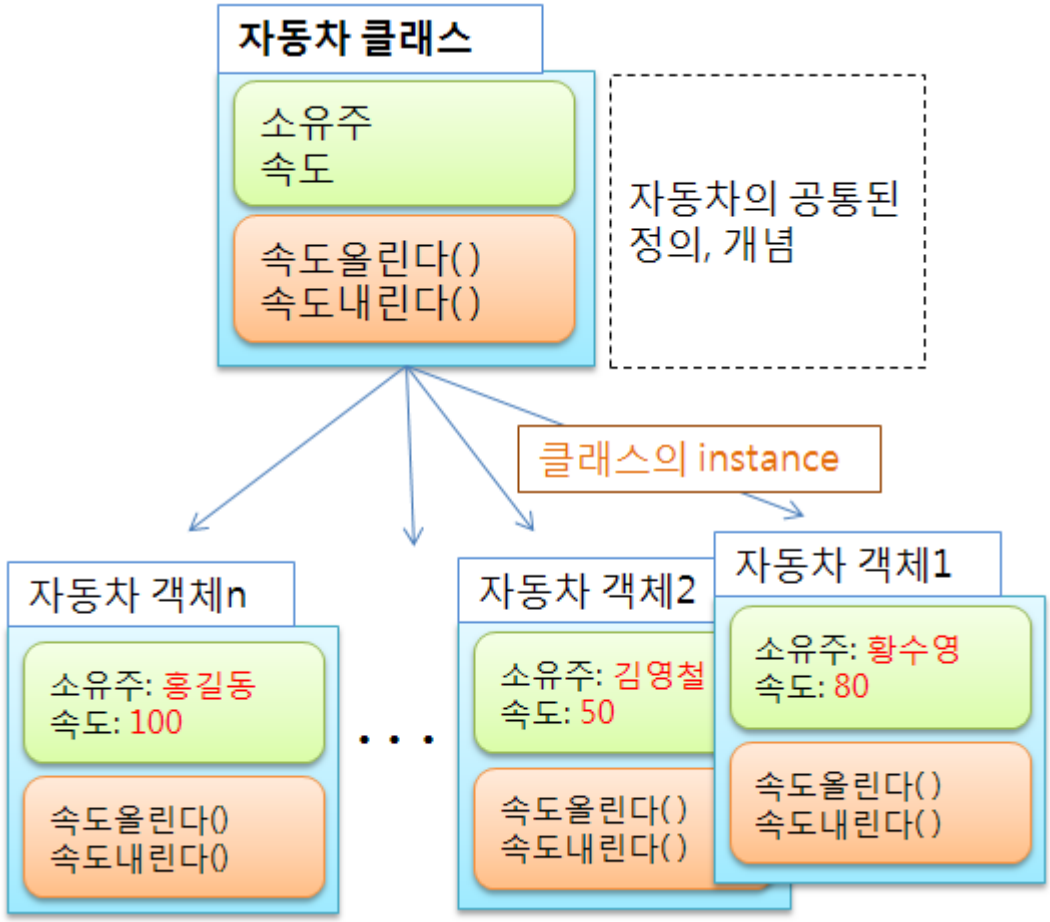
인스턴스와 객체가 같다고 알고 있어도 무리는 없습니다.

클래스의 인스터스를 객체라고 하니까요!

2, 클래스(class)

클래스(class)란 **상태와 그 상태(데이터,속성)를 다루는 행동들(메소드,연산)의 집합**입니다.
또 클래스는 객체들의 **공통된 개념을 표현하고 정의**합니다.
그래서 클래스는 객체의 **틀(template)**이라고도 합니다.
많은 책들의 정의는 달라도 의미는 같습니다.

객체지향에서 '클래스'와 '객체'는 꼭 구분하고 사용해야 합니다.
클래스는 개념이고 정의이며 객체는 클래스의 인스턴스(실체)입니다.

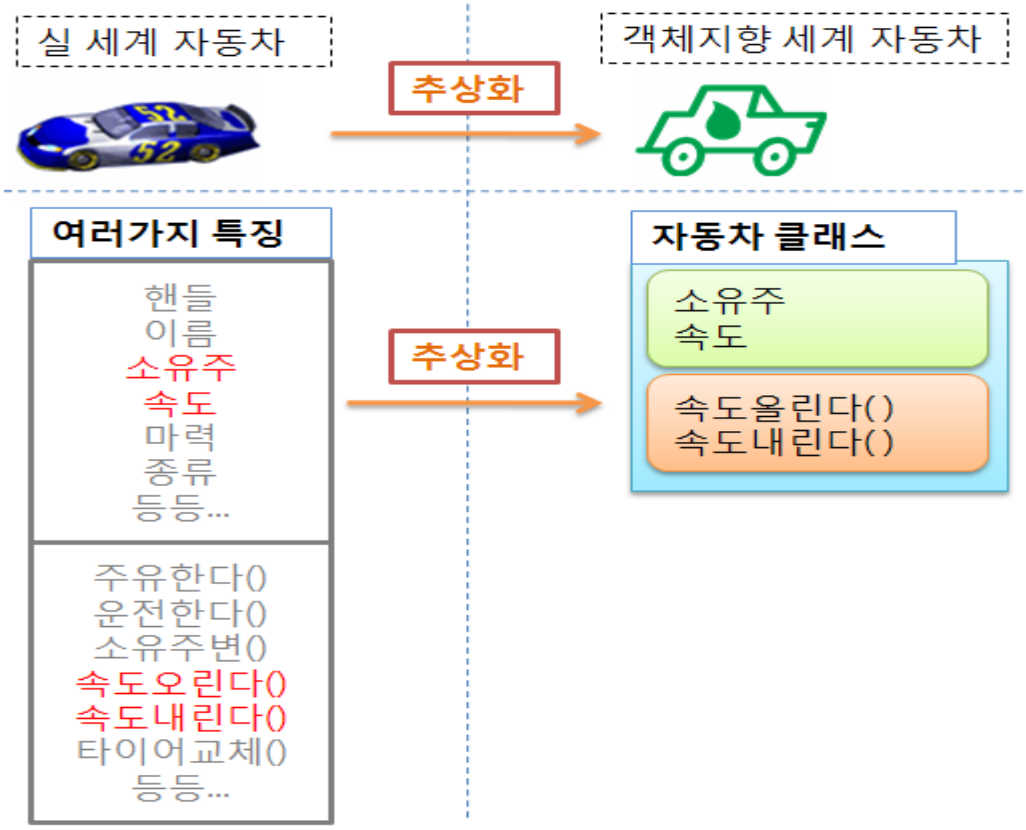


3, 추상화(abstraction)

추상화란 공통된 요소나 중요한 기능을 간추려 내는 작업을 의미합니다.
객체지향에서 추상화는 아주 중요한 요소 중 하나입니다. 추상화를 잘하느냐? 못하느냐?는 곧, 좋은 프로그램이냐? 나쁜 프로그램이냐?로 분류되기도 합니다.

우리가 앞으로 만들어 내는 클래스는 모두 추상화를 통해 만들어지게 됩니다.
프로그램은 사람들의 문제를 효율적이고 재미있게 해결하기 위해 존재합니다. 그러다 보니 프로그램은 사람들의 문제를 가상화(컴퓨터로 옮겨 놓다)했다고 보시면 됩니다. 이 같은 프로그램은 사람들의 문제를 컴퓨터로 옮겨 표현하기 때문에 옮기고자 하는 사항들을 간추려 내고 뽑아내기 위한 작업이 필요합니다. 한마디로 필요한 것들과 필요하지 않은 것들, 중요한 것들과 중요하지 않은 것들을 간추려 내야 합니다.
이렇게 중요한 내용과 그렇지 않은 내용, 공통된 내용을 선별하고 선택해 내는 작업을 추상화라 합니다.
이것저것 간추려 내는 작업에는 정답이 없습니다. 주어진 상황과 문제에 따라 얼마든지 달라지니까요.
그래서 추상화가 어렵고도 중요한 것입니다.

객체지향은 객체들의 세상입니다. 객체들은 클래스로 만들어집니다.
그래서 객체지향에서는 주어진 문제를 모두 클래스화 하기 때문에 추상화가 곧 클래스 화입니다.
주어진 문제를 추상화하여 어떤 클래스로 표현할 것인가? 클래스는 어떤 상태와 행동을 하도록 할 것인가? 클래스 간의 관계는(상호작용) 어떤 것들이 있는가?
이런 일련의 작업들이 추상화 즉 클래스화입니다.



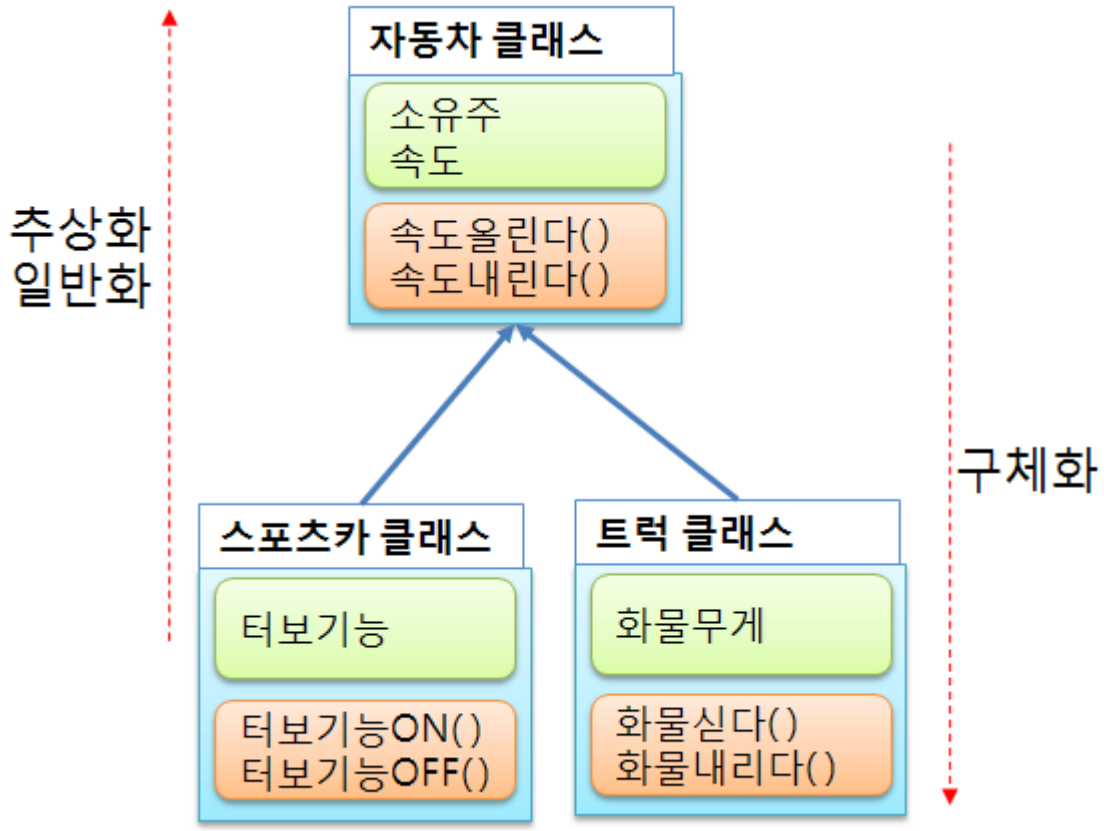
프로그래밍에서 추상화한 형식을 ADT(Abstract Data Type)이라 합니다.
사용자가 정의했다고 해서 UDT(User Definition Type)라고도 합니다.
그래서 객체지향은 클래스를 ADT 나 UDT 라 합니다.

객체지향의 클래스는 'ADT + 알파(다형성, 상속등)' 정도로 알아 두시면 됩니다.
사실 위 내용이 약간씩 차이가 있지만 그냥 같다고 보셔도 무방합니다.

어렵죠? 어렵습니다. 지금 이해가 힘들면 다른 공부를 병행하며 꼭 복습하여 알아둬야 합니다.

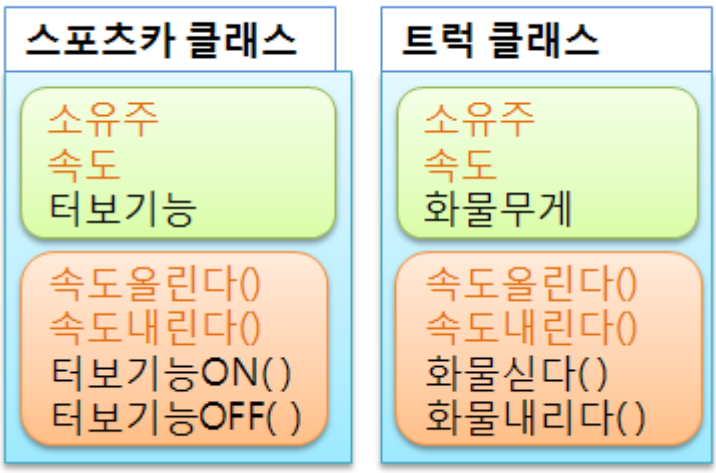
4, 상속(inheritance)

객체지향(OO)의 가장 중요한 세 가지 특징은 상속, 캡슐화, 다형성입니다. 이제 이 세 가지를 배워 보도록 하겠습니다.
상속은 하위 클래스가 상위 클래스의 모든 특성(속성과 행동(메소드))을 이어받는 것을 말한다.
OO 에서 상속은 코드 재상용의 가장 기본적인 내용이다.
하나의 클래스 자동차 클래스가 있을 때 또 다른 클래스 스포츠카와 트럭 클래스는 자동차 클래스의 **모든 속성(소유주, 속도)과 행동(속도올린다, 속도내린다)를 이어받습니다.**



스포츠카 클래스나 트럭 클래스는 좀 더 일반적이고 추상적인 자동차클래스에 속하며
스포츠카도 자동차이므로 속성 소유주와 속도를 가지며 행동인 속도 올린다와 속도 내린다를 갖습니다.
트럭도 자동차이므로 속성인 소유주와 속도를 가지며 행동인 속도 올린다와 속도 내린다를 갖습니다.
이렇게 스포츠카와 트럭은 자동차의 모든 특성인 속성과 행동을 상속받게 됩니다.

아래는 자동차 클래스 상속 후 스포츠카와 트럭 클래스가 가지게 되는 속성과 행동입니다.



상속에서 추상적인고 일반적인 클래스를 상위 클래스(super class), 부모 클래스(parent class)라하며
구체적인 클래스를 하위 클래스(sub class), 자식 클래스(child class)라 합니다.

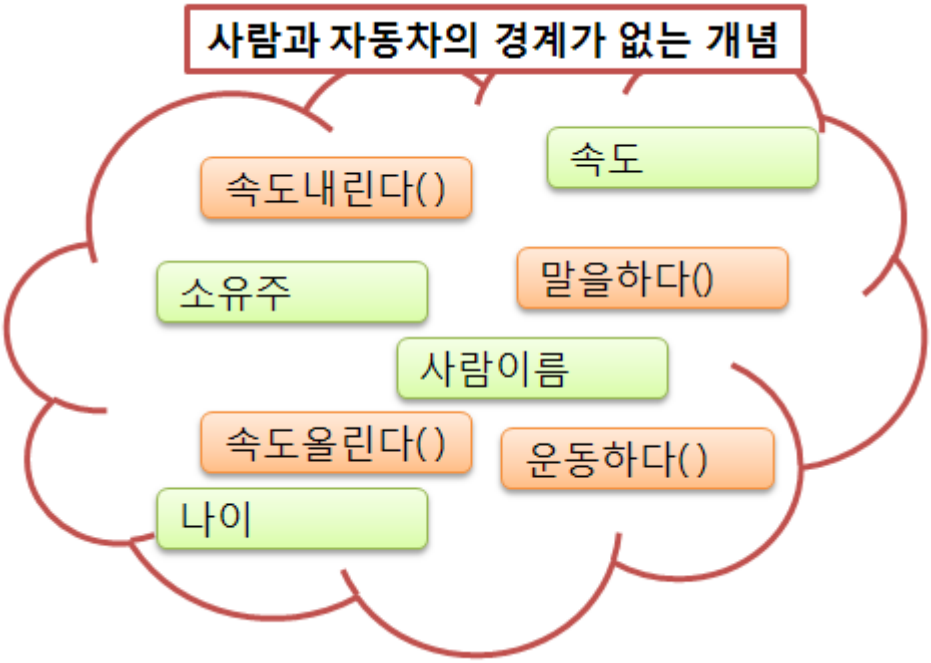
5, 캡슐화(encapsulation)

캡슐화는 크게 두가지 개념을 가지고 있습니다.

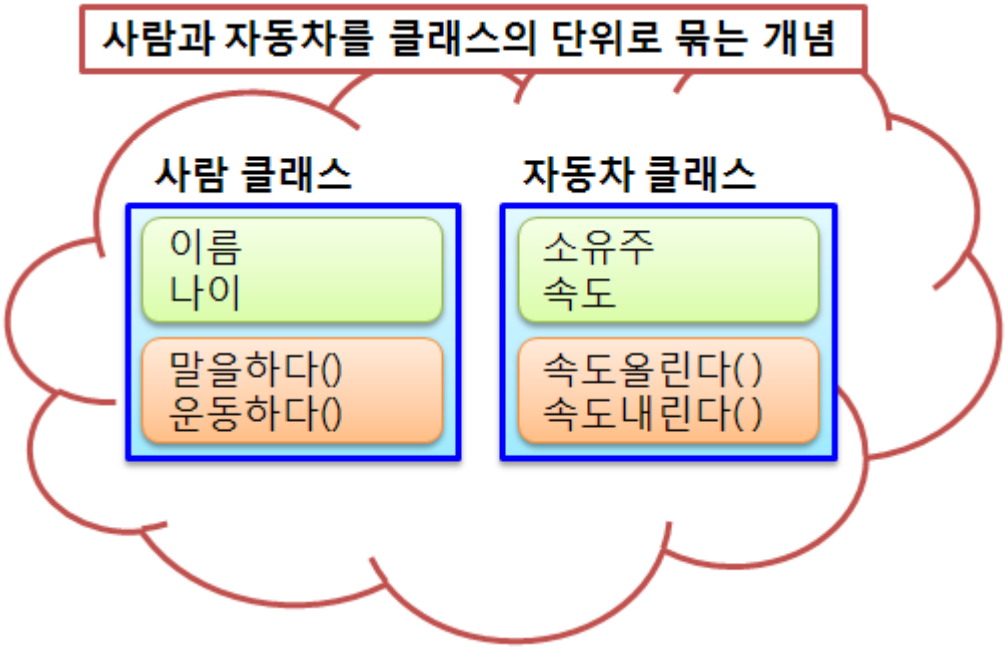
- **첫째, 하나의 단위로 묶는 개념**
- **둘째, 정보은닉(information hiding)의 개념**

캡슐화와 정보은익을 다르게 보는 시각도 있지만 우리는 캡슐화를 위 두가지로 보겠습니다.

첫째, 하나의 단위로 묶는 개념입니다.
객체는 속성과 행동을 가지고 다른 객체와 고유하게 구분되어 독립적으로 존재한다고 했습니다. 이때 속성과 행동을 묶어 다른 객체와 고유하게 구분하는 개념이 캡슐화입니다.
간단한 예로 아래 차동차와 사람이라는 속성과 행동이 있을 때



속성과 행동을 공통되고 유사한 개념을 갖는 클래스로 묶습니다.



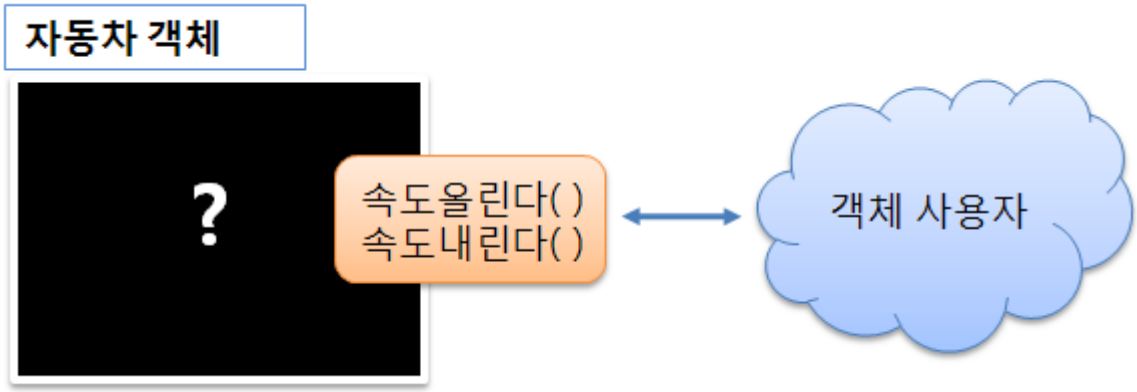
둘째, 정보은닉(information hiding)의 개념입니다.

객체의 불필요한 세세한 내용은 객체 사용자에게 노출 시키지 않는 개념입니다.

객체는 속성과 행동으로 이루어지므로 이 객체의 속성과 행동에서 객체 사용자에게 꼭 필요한 것만 노출(공개)하는 것을 말합니다.

예로 자동차 객체를 사용하는 객체가 속도를 올리고 내리는 것 외에 알 필요가 없다면 자동차 객체는 두 행동(속도 올린다, 속도 내린다) 외엔 모두 감추게 됩니다.

그래서 **객체를 행동들만의 집합**으로 보고 프로그래밍하는 경향이 있습니다. 이러면 프로그래밍이 더 쉽고 명확해집니다. 사실 객체는 속성과 행동들의 집합이지만요.



자동차 객체를 사용하는 클라이언트 입장에서는 자동차 객체의 세세한 내용을 알 필요 없이 자동차 객체와 소통(통신)하므로 자동차 객체를 사용하고 다루기가 쉬울뿐더러 자동차 객체도 객체 자신을 변경하고 확장하기도 쉽습니다.
이때, 행동인 속도 올린다, 속도 내린다 만을 통해서 자동차 객체와 소통(통신)하고 있으므로 이 행동이 객체 간 인터페이스가 됩니다. 인터페이스의 자세한 내용은 아래에서 공부합니다.

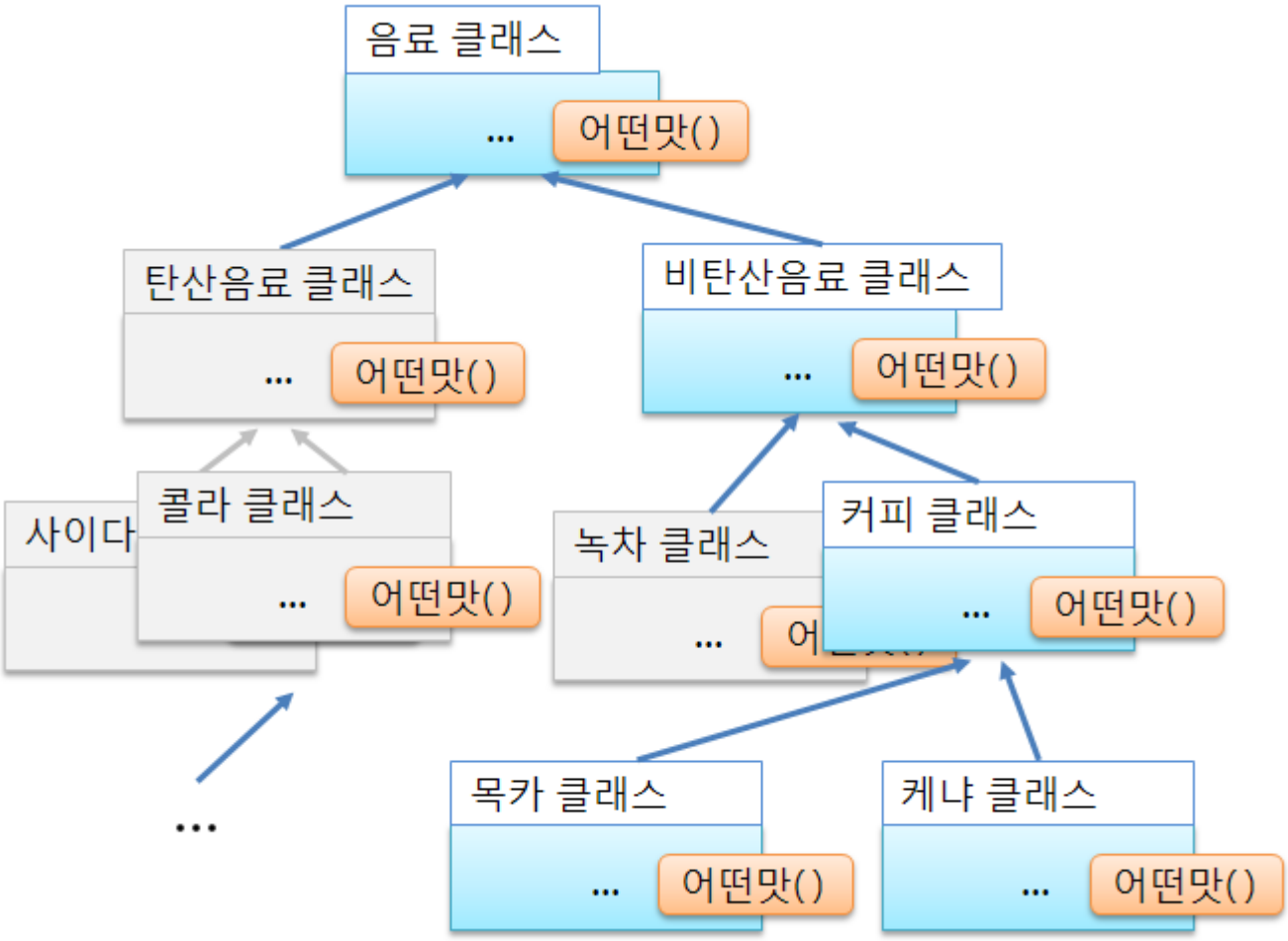
6, 다형성(polymorphism)

다형성은 상호 관련된 객체들을 동일한 방식으로 다루는 개념을 말합니다.
다시 말하면 같은 메시지에 대해 자신의 객체가 어떻게 반응해야(행동해야) 하는지 아는 것을 의미합니다.
또다시 말하면 메소드 호출 시 정확한 객체의 메소드(행동)가 호출되는 것을 말합니다.

어렵다. ㅠ.ㅠ... 다형성이 처음에 가장 이해하기 어렵습니다. 하지만, 가장 중요한 개념 중 하나입니다.

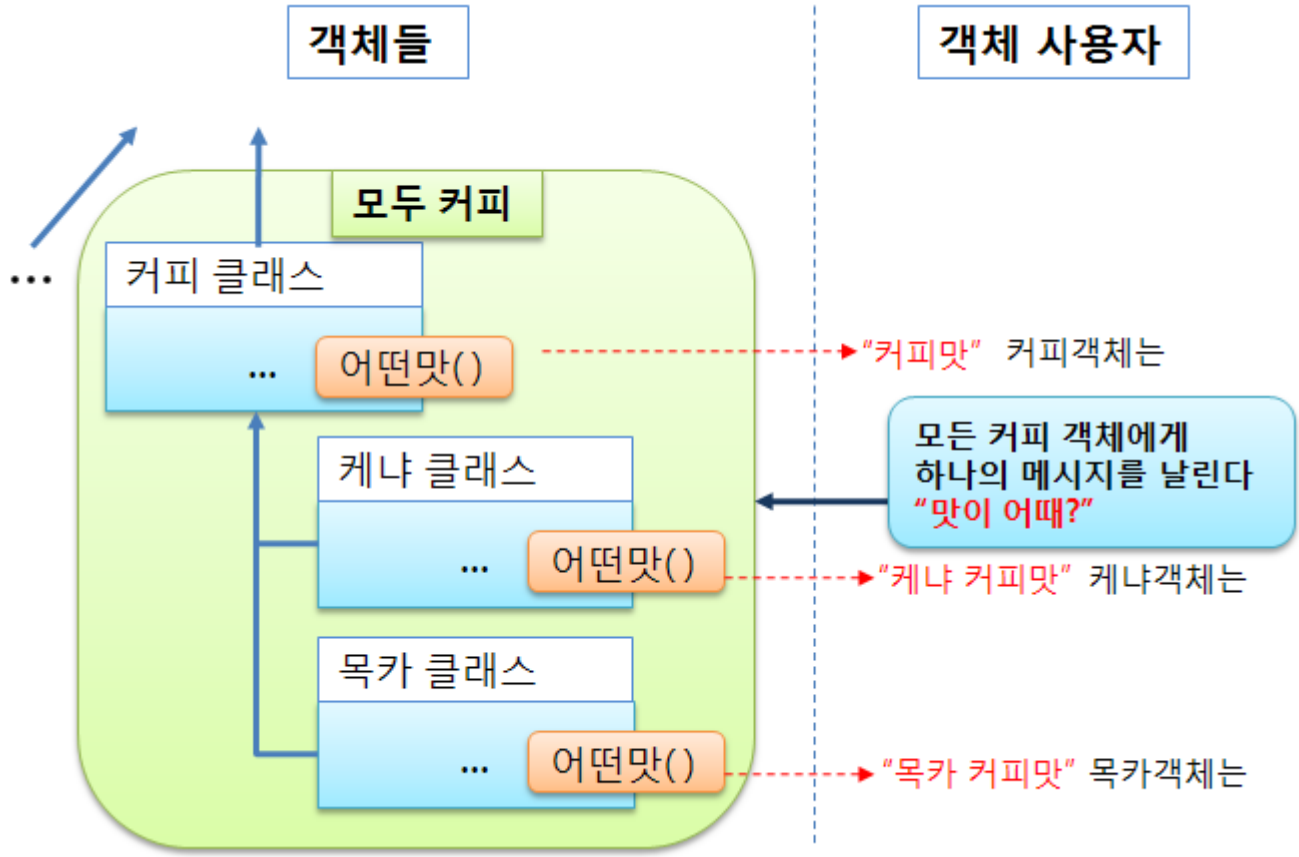
다형성은 클래스 다형성과 임시적 다형성이 있는데, 언어 대부분에서 클래스 다형성만을 지원하므로 우리도 클래스 다형성만을 생각합니다.

아래 그림처럼 계층구조를 갖는 음료 클래스가 있다고 가정합니다.
음료의 가장 중요한 요소는 맛이므로 모든 음료 클래스는 맛을 응답할 수 있는 메소드를 가지고 있습니다.



음료 중 커피 계층구조만 확대해 보면 아래와 같습니다.

이때, 커피에게 맛을 물어보면(메시지를 보내면) 각 클래스의 객체마다 자신의 객체에 맞는 응답(메소드 호출됨)을 합니다. 커피 객체는 "커피맛'이다, 케냐 객체는 "케냐 커피맛"이다, 목카 객체는 "목카 커피맛"이다고 응답합니다. 이렇게 **관련된 객체들(커피 객체들)**에게 **하나의 메시지("맛이 어때?")**를 보내면 **자신의 객체에 맞게 응답(메소드가 호출됨)**하는 것을 **다형성**이라 합니다.

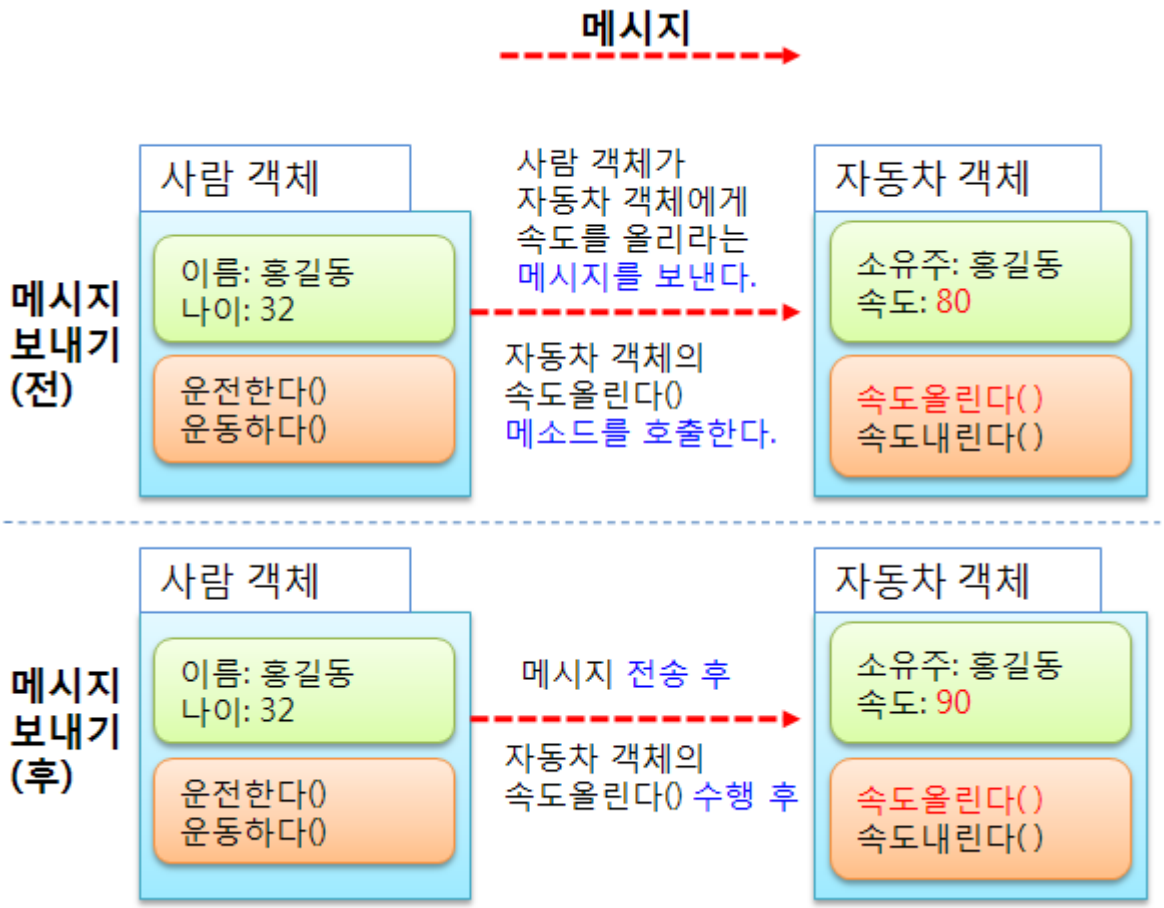


메시지는 아래에서 자세히 공부합니다.

7, 메시지(message)

객체는 다른 객체와 상호 작용(통신, 소통)한다고 했습니다. 그렇다면, 객체들은 어떻게 상호 작용하는 것일까요?
바로 메시지를 사용하여 상호 작용하게 됩니다.

예로 사람 객체가 자동차 객체에 메시지를 보낸다는 의미는 사람 객체에서 자동차 객체의 메소드를 호출하는 것을 말합니다.
만약, 홍길동이라는 사람 객체가 자동차 객체의 속도를 올리려고 메시지를 보내면 자동차 객체의 '속도올린다()'메소드를 호출하고 '속도올린다()'메소드의 기능을 수행하며 자동차 객체의 속도가 올라갑니다.
아래 그림 참고.



8, 인터페이스(interface)

인터페이스는 큰 의미로 두 객체 간의 소통 역할을 하는 중계자를 의미합니다.
예로 지방과 지방을 연결하는 '도로', 강으로 나뉜 지역과 지역을 연결하는 '다리', 매도자와 매수자를 소개하는 '공인 중개사'가 모두 인터페이스입니다.

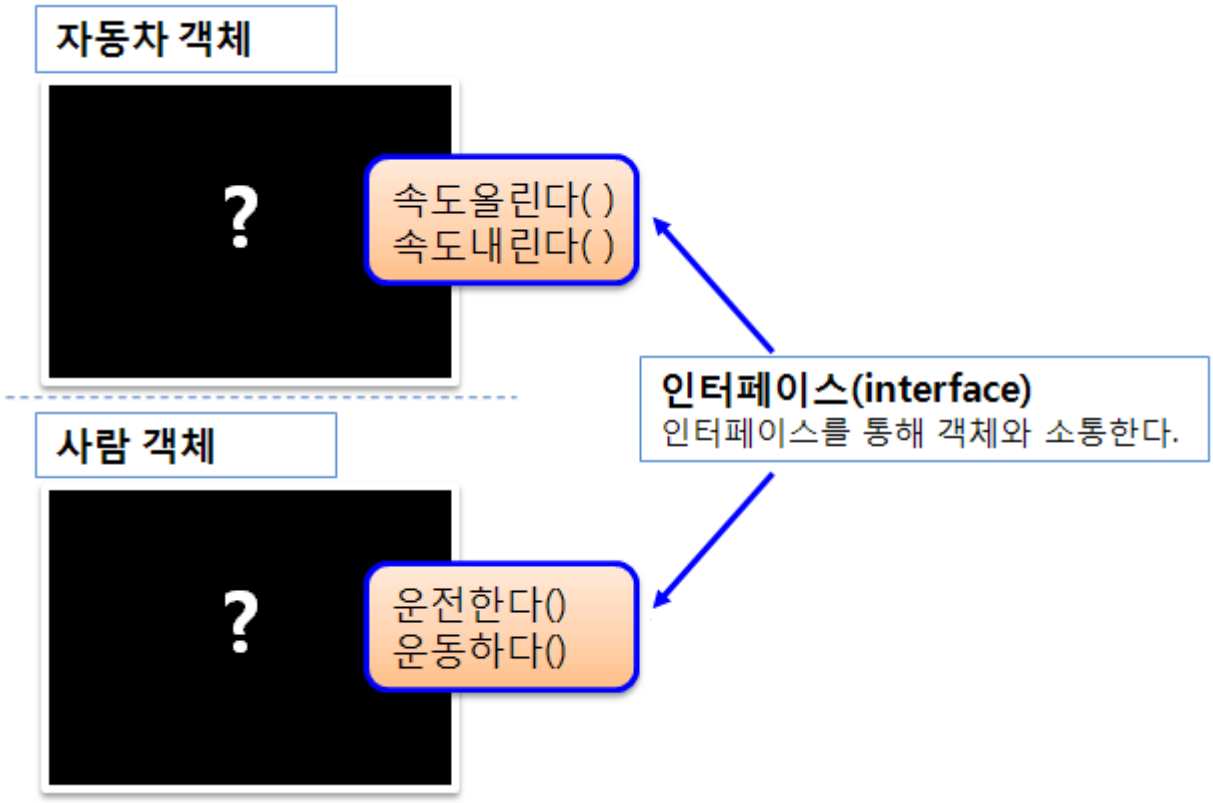
객체지향에서 인터페이스는 객체의 내부와 외부, 서버와 클라이언트 사이의 중계자 역할을 하는 놈을 말합니다.
그렇다면, 이 중계자 역할을 하는 놈은 무엇일까요?
바로 외부로 공개된 메소드들입니다. 캡슐화를 공부할 때 객체에서 외부로 공개된(노출된) 것은 필요한 메소드라고 했습니다.
객체지향은 항상 공개 메소드를 통해 객체와 통신하게 됩니다.
다시 말씀드리지만 그래서 객체가 속성과 행동들의 집합이지만 **객체를 행동(공개 메소드)들만의 집합**으로 보는 경향이 강합니다.

엄밀히 말하면 메소드의 기능은 객체 외부로 공개(노출)되지 않습니다. 외부에서는 메소드의 기능을 볼 수 없습니다.

외부로 공개된 것은 메소드의 시그니처(signature)입니다.

메소드의 시그니처는 함수의 반환형, 이름, 매개 변수 자료형을 의미합니다.

정확히 말하면 인터페이스는 공개된 메소드의 시그니처입니다.



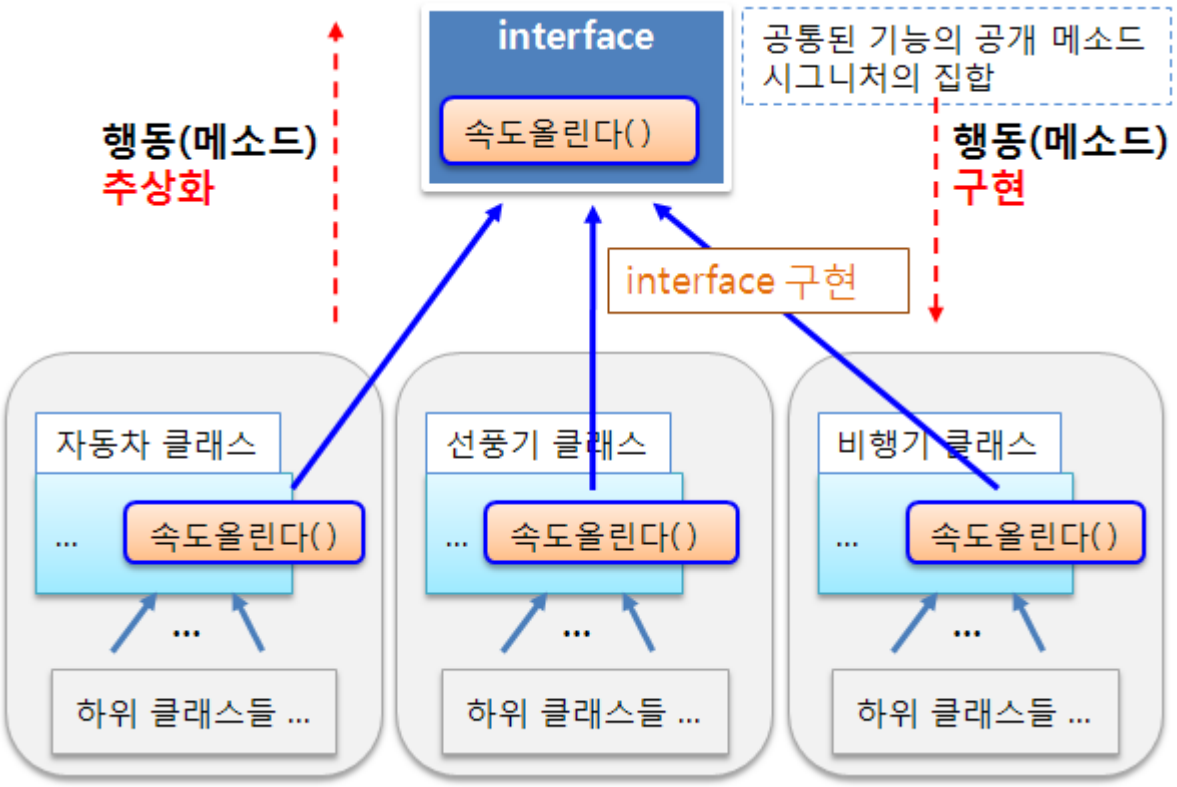
하나 주의할 것은 **많은 객체 지향 언어에서 인터페이스 형식(언어 문법)을 제공**한다는 것입니다. 인터페이스는 객체지향의 중요한 요소 중 하나이므로 언어 자체에서 지원하여 보다 쉽고 효율적인 객체지향 프로그램을 만들 수 있도록 합니다.

언어에서 지원하는 인터페이스가 무엇인지 보도록 하겠습니다.

Java 에서도 인터페이스 형식을 지원합니다.

언어에서 지원하는 인터페이스는 단지 공통된 기능을 하는 공개 메소드 시그니처(signature) 집합입니다.

다시 말해 **인터페이스는 클래스들의 공통된 메소드 시그니처 집합(signature)**입니다.



위 그림과 같이 자동차 클래스도 '속도올린다()'라는 기능을 가지며 선풍기 클래스도 '속도올린다()'라는 기능, 비행기 클래스도 '속도올린다()'기능을 가지므로 '속도올린다()'라는 행동을 인터페이스로 추상화합니다.
그리고 이 인터페이스를 갖는 모든 클래스는 '속도올린다()'라는 기능을 구현하므로 이 인터페이스를 구현한 모든 객체에 같은 메시지를 보낼 수 있습니다.

즉, 클래스에 무관하게 서로 다른 객체들을 하나의 인터페이스로 기능을 수행할 수 있다는 것입니다.