# OOP에 대하여

Object Oriented Programming의 약자인 OOP는 절차 지향 프로그래밍과 같은 프로그래밍 기법의 한 패러다임이다. 한글로 객체지향 프로그래밍이라고 불리우는 이 프로그래밍 기법은 기존의 절차 지향 프로그래밍과 달리 하나의 데이터를 처리하기 위해서 실행 순서대로 프로그래밍 하는 것이 아니라 데이터를 객체화 하고, 루틴들을 하나의 독립된 객체로 설정하며 프로그램의 구성하는 기법이다. 이를 통하여 코드는 재사용성의 증가라는 큰 이점이 생기고 이로인하여 관리자의 유지/보수/관리가 용이해진다는 장점이 있다.

## 객체란 (object)

객체란 말 그대로 대상을 나타내는 단어이다. 여기서 대상이란 사람처럼 물리적인 실체가 있는 것들 뿐만 아니라, 수학, 철학처럼 논리적이고 추상적이지만 설명 가능한 대상도 가능하다.

객체는 데이터와 기능으로 묶어 설명이 가능한데, 이 때 데이터를 프로퍼티라 하며, 기능은 메서드라고 칭한다. 메서드 또한 프로퍼티의 일종으로써 특정한 data대신 동적인 기능인 함수가 존재 함으로 특별히 함수가 있는 프로퍼티를 Method라 칭한다. 이 때 property는 key 와 value로 이루어져 있으며 method또한 value에 function을 담고 있는 것 말고는 생김새가 동일하다.

특별히 객체는 인스턴스(instance)라 불리기도 하는데, 이는 보통 객체가 어느 클래스에서 정의되어 실체화 되었을 때 주로 사용되는데, OOP관점에서 인스턴스는 객체가 메모리에 할당되어 실제 사용되었을 때 ‘인스턴스’라 칭한다. 즉 인스턴스는 객체와 클래스의 관계로 한정 지어서 사용할 필요는 없다.

## 클래스 (class)

클래스는 같은 특징을 가진 객체의 형태를 가져 특정한 모습을 가진 객체를 복제할 수 있는 틀이다. 그 목적 또한 객체를 생성하는 것이며, 전달인자를 통하여 객체 마다 초기 모습이 약간씩 다를 수 있게 하는 기능을 가졌다.

## 추상화 / 캡슐화 / 다형성 / 상속성

객체지향 프로그래밍을 구성하는데 있어 4가지의 요소를 기반으로 프로그래밍을 진행한다.

## 추상화(Abstraction)

OOP를 다루는 프로그래머에게 OOP의 목적이 무엇이냐고 물어본다면 이렇게 대답 할 수 도 ! 있다. (논란의 여지가 있는 말이므로 참고만 하자) 바로 현실 세계의 법칙을 적용하여 코드를 구성하는 것이라고 말이다. 객체란 러닝 자바스크립트의 이선브라운이 말했던 것처럼 논리적인 기능과 데이터를 묶어 놓은 것이다. 그렇기에 우리가 현실 세계에 있는 물체를 정확히 정의하고 기능의 구현 방법을 명확히 명시한다면 현실 세계의 물체가 물질이나 추상적인 개념들을 객체화 할 수 있다. 그러나 이 모든 복잡한 개념들을 한 Class에서 전부 정의하기란 불가능 혹은 매우 어렵다.

그래서 이 추상화라는 개념은 다음과 같은 특성을 객체의 클래스에 적용하여 클래스의 부피를 줄인다.

1. 대상의 특성 중 불필요한 부분을 무시하고 필요한 공통점만을 다룬다.
2. 현실의 복잡성을 극복하고 객체의 존재 목적에 집중한다.
3. 사물들 간의 공통점은 취하고 차이점은 버리는 일반화를 통한 단순화
4. 중요한 부분의 강조를 위해 불필요한 세부 사항을 제거하는 단순화

결국 추상화는 한 마디로 현실세계에서 코드상으로 객체를 옮길 때 얼마나 단순화 시켜서 가져올 것인가에 대한 문제로 이루어진다. 그렇기 때문에 추상화는 문제 영역과 관점에 의존적이며 특정 코드에서 필요 없는 데이터와 기능이 다른 영역의 코드에서는 추가 되어있을 수 있다.

## 캡슐화 (Encapsulation)

캡슐화는 일반적으로 연관 있는 변수와 함수를 클래스로 묶는 작업을 일컫는다. 즉 데이터 구조와 데이터 다루는 방법들을 결합시켜 특정 객체의 특징을 클래스로써 구체적으로 사양을 명세하고 그 작동 원리를 기재하는 작업이라고 할 수 있다.

이 캡슐화는 은닉성과 혼용해서 사용되기도 한다. 은닉성은 외부로 객체가 들어내서는 안되는 함수와 변수들을 숨기고 객체 내부에서 겉으로 들어나는 함수를 통해서 제어가 되는 특징을 일컫는다. 그렇기에 엄연히 말해서 캡슐화를 통한 객체의 특징을 명세하는 과정에서 특정 함수나 변수의 은닉화가 진행되는 것이다. 결국 이러한 캡슐화를 통하여 객체는 외부의 잘못된 사용으로 인하여 손상을 입지 않게 끔 보호된다.

Java와 c#같은 언어에서는 접근 제한자(Access Modifier)를 통하여 함수와 변수를 은닉화 하지만 javascript와 c같은 언어에서는 접근 제한자가 따로 없으며 이 같은 경우 들어나는 함수와 변수의 스코프를 조정하여 클로저를 생성한 다음 은닉화를 진행한다.

## 상속성(Inheritance)

상속이란 부모가 자식에게 특징을 물려주어 자식에게도 동일한 특징이 존재하는 동작을 의미한다. 이 같은 경우 현실에서는 부모가 자식을 특정하여 특징을 물려주는 행위이지만, OOP에서는 자식이 부모를 선택한다. 예를 들어 아래 코드에서는 Dog 클래스에서 Mammalia(포유류)를 선택하는 모습을 볼 수 있다.

class Mammalia{

pregnant(){

return Object.assign({},this);

}

}

class Dog extends Mammalia{

beak(){

console.log("bow bow")

}

}

Mammalia와 같이 어떤 클래스의 상위 클래스를 parent class 부모 클래스라 부르며, Dog과 같이 어떤 클래스를 상속 받는 클래스를 child class 자식 클래스라 부른다. 이때 부모 클래스와 자식 클래스는 상대적인 개념이며, 어떤 클래스를 어떤 상황에서 호칭하는지 여부에 따라 다르게 호칭해야 한다.

이와 같이 상속은 자식 클래스가 부모 클래스의 변수와 함수를 물려 받는 것이며, 이때 하위 클래스에서는 부모 클래스에서 지정한 특정 변수와 함수를 사용하기 위해서 부모 클래스의 생성자 함수를 호출해야 될 때도 존재한다.

이러한 상속의 효과는 다음과 같다.

1. 반복된 코드를 줄여줌으로써 자식 클래스의 구현이 빨라진다.
2. 코드의 재사용성이 증가한다.
3. 유지 보수가 편리해진다.
4. 현실 세계의 객체간 관계를 구현하기 용이하다.

위의 상속의 효과에서 우리는 4번에 조금 주목할 필요가 있다. 4번에서 나타나는 문제는 바로 다중 상속 (Multiple inheritance)이다. 다중 상속이란 어떤 클래스가 하나 이상의 상위 클래스로 부터 여러가지 행동이나 특징을 상속 받을 수 있는 것을 말한다. 이 다중 상속을 예를 들면 부 A와 모 B가 존재한다. 이때 자식 C는 부 A와 모 B의 특징을 물려 받으려 하지만 부 A와 모 B의 동일한 이름의 동작 메커니즘이 전혀 다른 기능인 소변보기라는 메서드를 물려받게 된다. 이때 자식 C는 부 A와 모 B중 어느 메서드를 물려 받아야 하는가라는 문제에 직면하게 된다. 이와 같은 문제는 컴퓨터에게 혼란을 가중시켜 주기에 현재 많은 언어가 다중 상속을 지원하지 않게 된다. 그렇지만 이와 같이 다중 상속과 동시 상속이라는 문제는 real world를 구현하려면 꼭 해결해야 하는 문제이다. 다양한 언어들은 각기 다른 방법으로 위의 문제를 우회하였고, javascript에서는 믹스인(mixin) 이라는 방법으로 해결하였다.

Mixin 기법은 단순히 부모 A와 부모 B의 클래스가 모든 기능을 가지고 있는 거대한 클래스를 생성하여 다중 상속을 동시에 받는 것이 아니라, 특정 기능을 가지고 있는 클래스를 만들어 동시 상속을 구현 하는 방법이다.

class Machine{

}

function FlyMixin(mixClass){

return class extends mixClass{

fly(){

console.log("go")

}

};

}

function DriveMixin(mixClass){

return class extends mixClass{

drive(){

console.log("go")

}

};

}

class FlyCar extends DriveMixin(FlyMixin(Machine)){

}

(javascript Mixin예제)

상속성을 공부하며 자주 들을 수 있는 단어로 **오버라이딩**(Overriding)이란 단어가 존재한다. 오버라이딩은 부모의 클래스로부터 물려받은 메서드와 변수를 자식 클래스에서 다른 기능을 가진 메서드로 재정의하여 사용하는 것을 말한다.

## 다형성(Polymorphism)

다형성은 같은 클래스의 인스턴스들이 같은 형태일 수 있지만 그 객체를 만드는 방법에 따라 객체의 형태가 다른 특징을 말한다. 예를 들어 붕어빵을 제조할 때 붕어빵 틀에 붓는 재료에 따라서 피자 붕어빵이 되기도 하고, 팥이 들어있는 일반 붕어빵이 될 수 있다. 이러한 특징으로 인하여, Class의 재 사용성이 대폭 증가한다. 즉 그 어떤 붕어빵이든 본디 모습은 붕어빵이기에 어떤 모습을 하더라도 붕어빵이게 되는 것이다.

단형성을 가진 프로그래밍 언어에서는 어떤 값을 문자열 형식으로 변환하는 경우 개별적인 함수를 만들어 각 type들을 변환해야 한다. 그러나 다형성을 가진 언어는 미리 범용적인 메서드를 지정함으로써 객체의 종류와 상관없는 추상도가 높은 변환 형식을 구현 할 수 있다.

*//숫자를 문자열로 바꾸는 경우*

string = StringFromNumber(number);

*//날짜를 문자열로 바꾸는 경우*

string = StringFromDate(date);

(단형성을 가진 프로그래밍 언어의 경우)

*//숫자를 문자열로 바꾸는 경우*

string = number.StringValue();

*//날짜를 문자열로 바꾸는 경우*

string = date.StringValue();

(다형성을 가진 프로그래밍 언어의 경우)

특히 java의 경우 오버로딩(Overloading)은 OOP에서 다형성을 지원해주는 역할을 한다. Overloading은 같은 이름의 메서드를 만들 수 있도록 하고, 이 메서드는 각각이 자료형이 다른 전달인자를 받음으로써 기능을 달리 하는 것이 아니라 각각의 오버로딩된 메서드에서 처리만 다르게 하고, 결과는 동일케 한다.

public class Parse{

public dateToString(int date){

...

}

public dateToString(Date date){

...

}

public dateToString(float date){

...

}

}

(java 언어의 오버로딩)

## 과연 OOP는 절대적이고 완벽한 패러다임인가 ?

OOP는 절차지향프로그래밍 보다 훨씬 코드의 재사용성 증가로 인하여 개발시간 단축, 좀 더 정확한 코딩을 보증하지만 코드의 난이도가 상승한다. 특히 다중 상속을 고려하게 되는 경우 클래스간 상속 받는 데이터와 함수의 상속 순위를 지정해야 하는데 이로 인하여 클래스를 일관되게 작성하기 어려워진다. 그래서 아예 다중 상속을 지원하지 않는 프로그래밍 언어도 존재하고, 다중 상속을 javascript의 mixin기법과 interface 같이 우회하기도 한다.

캡슐화를 통해 은닉된 데이터는 보통 module pattern이라고 불리는 getter와 setter를 사용하는데, 이는 private한 성질의 데이터를 외부에서 사용하고 할당하기위해서 가지는 패턴이다. 그러나 이 같은 패턴 형식은 한개의 데이터나 함수의 용도에 따라서 코드의 양이 2배로 증가하는데 기여한다. 만약 private한 성질의 데이터가 50개면 이를 위한 get set 메서드만 100개가 되어 버리는 일이 발생한다. 결국 getter와 setter를 사용하는 과정이 복잡해지고 너무 많아지게 되면 프로그래머는 피로감에 의하여 모든 변수를 public으로 설정하여 캡슐화가 깨져버리는 일이 발생하게 되어 OOP의 의미가 퇴색되어 버린다.

마지막으로 C같이 아직 많이 사용되나 절차지향언어에서 OOP를 구현하는 것은 상당한 인내를 요구한다. 물론 C++이라는 대안이 있지만 이는 새로운 언어의 학습이라는 프로그래머에게 상당한 부담을 안겨준다. 임베디드나 펌웨어 관련 프로젝트에서는 여러가지 사정으로 C언어를 사용하여 큰 규모의 소스코드를 작성해야 하는 경우가 존재하는데, 이런 경우 절차 지향 방법으로 진행하는 경우 프로젝트 기간이 한도 끝도 없이 길어지게 된다 이런 경우 OOP 설계를 도입하여 모듈방식으로 코드의 혼잡을 피해야만 한다. 따라서 위와 같은 프로그래머들은 객체지향적인 C의 설계는 반드시 익혀야 할 내용이다. (물론 이런 프로그래머는 초 고급 인력에 속하며 인건비가 상당하다.)

## OOP의 대안

최근 OOP의 대안으로 뜨고 있는 프로그래밍 패러다임은 함수형 프로그래밍이다. 함수형 프로그래밍은 순사한 함수를 작성하고, 공유된 상태와 변경 가능한 데이터 및 부작용을 피하여 소프트웨어를 작성하는 프로그래밍 기법이다. 함수형 프로그래밍과 OOP의 대조적인 부분은 특정 상태는 객체의 메서드로 전달되는 것이 아니라 순수한 함수를 통해 전달 되는 것이다. 즉 Side effect를 없애고, 파라메터로 전달인자를 받는 것으로 인하여 외부의 값에 접근하지 않아야 한다. 이 같은 함수형 프로그래밍의 장점은 OOP나 PP코드 보다 훨씬 간결하고 예측이 쉬워지게 되어 테스트에 용이하다.

## 참조 문건

<https://velog.io/@stampid/OOP>란

<https://victorydntmd.tistory.com/117>

<https://goodgid.github.io/What-is-OOP/>