20170118
객체 지향 언어의 특징에 관하여

1.추상화란 무엇인가?
객체 지향 프로그래밍에서 일반적으로 추상 자료형을 클래스, 추상 자료형의 인스턴스를 객체, 추상 자료형에서 정의된 연산을 메소드, 메소드의 호출을 메시지라고 한다.
클레스 구현에 있어서 추상화는 “매우 간단히” 말해서 클래스로 구현하고자 하는 것에서 최고로 중요한 것 만을 서술하는 요약이라고 할 수 있다. 예를 들어, 게임이나 소셜미디어 애플리케이션을 제작할 시에 클래스로서 구현해야 할 ‘소비자, 인간’ 이라는 클래스에 인간이 가지고 있는 모든 사항을 작성하고자 한다면, 수천만줄의 코드가 필요할 것이며, 이는 현실적으로 불가능할 것이다. 혹여나 가능하다 하더라도 개발자와 경영진이 원하지 않는 기능 구현을 위한 상당한 리소스 자원, 노동력이 소요될 것이다(예를 들어, RPG 게임의 클래스엔 ‘나이’라는 프로퍼티를 작성할 필요가 없다)

2. 캡슐화와 은닉화는 무엇인가?
객체는 속성(데이터)와 속성을 처리하는 메소드를 갖고 메모리에서 활동한다. 이때 객체를 사용하는 사용자 입장에선 객체의 기능만 알고 있다면, 객체가 어떻게 데이터를 처리해야 하는 지는 알 필요가 없다.
따라서 개발자는 클래스의 속성과 메소드에 접근 제한을 설정하여 외부로부터의 속성, 메소드로의 접근을 차단할 수 있다. 이때 제한의 등급은 자기 클래스 내부의 메소드에서만의 접근을 허용하는 Private, 자기 클래스 내부 또는 상속받은 자식 클래스에서의 접근을 허용하는 Protected, 모든 접근을 허용하는 Public으로 나눈다.
캡슐화란 데이터 구조와 데이터를 다루는 방법들, 즉 변수와 함수를 하나로 묶는 것으로, 객체가 맡은 역할을 수행하기 위한 하나의 목적을 한데 묶는 다는 아이디어에 기반해 이뤄져야 한다. 캡슐화에 성공하면 은닉화도 자연스럽게 효력이 나타난다.
또한 캡슐화는 코드의 수정없는 재활용을 위한 필수적인 개념이며, 이를 준수하지 않으면 객체 지향 언어의 다른 개념인 상속과 다형성은 성립이 불가하다. 이러한 캡슐화의 가장 좋은 예는 바로 클래스라고 할 수 있다.

3. 상속이란 무엇인가?
상속은 부모 클래스가 가지고있는 모든 것을 자식 클래스가 물려받아 같이 공유하며 나아가 확장(extends) 하는 개념이다. 부모 클래스를 상위 클래스(superclass)로 부르며 상속받는 자식 클래스를 하위클래스(subclass)라고 부른다. 부모 클래스에서 가지고 있는 추상적인 메소드를 자식클래스에서 구체적인 메소드로 오버라이드 할 수 있다.
상속의 장점은 다음과 같다.
첫째, 코드 재사용이 가능해진다. 새로운 종류의 subclass를 작성하면, superclass 에 이미 내장된 기능들을 사용할 수 있기 때문이다.
둘째, 계층구조를 표현, 즉 추상화를 구조화한다. 상속을 사용하면 공통적인 요소를 슈퍼클래스에 정의함으로써 일반화할 수 있다.
이에 대한 구체적인 예는 Xcode에서 활용 가능한 여러 명령어들이다. 이들 명령어들은 이미 상위 클래스에 구현되어 있어, 개발자는 상위 클래스에서 작성된 여러 프로퍼티, 메소드를 활용할 수 있다.
또한 개발자는 상속을 통해 파편화된 클래스에 서술된 프로퍼티, 메소드를 일일이 수정해야 하는 작업에서 벗어날 수 있다. 예를들어, RPG게임을 만든다고 했을 때, 만일 20명의 캐릭터, 캐릭터 별 5가지 프로퍼티를 설정해 작업을 한 다면, 개발자는 총 100번 이상 프로퍼티를 수정해야 하지만, 만일 각 캐릭터들의 상위 클래스로 2종족 클래스를 구성, 2종족의 프로퍼티를 5가지 설정해서 작업을 진행한다면 개발자는 약 10여번 정도만 프로퍼티를 수정하면 모든 캐릭터의 프로퍼티를 수정할 수 있다.

4. 다형성이란 무엇인가?
다형성이란 '여러 가지 형태를 가질 수 있는 능력'을 의미한다. 간단히 말해서 하나의 객체를 다양한 타입으로 선언하고 사용할 수 있다는 것을 의미한다. 다시 말해, 하나의 프로퍼티, 메소드 이름은 클래스(파일)마다 다른 ‘동작’을 개발자, 사용자에게 선보일수 있다. 이는 절차지향형 언어에서 하나의 프로퍼티, 메소드가 단 하나의 동작 만을 구사하는 것과 대조적이다. 예를 들어, 사용자가 점프 메소드를 사용한다고 가정하면, 이 점프 메소드는 캐릭터에 따라서 30cm 높이의 점프, 60cm 높이의 점프, 120cm 높이의 점프로 각기 다르게 구동될 수 있다. 만일 절차지향형 언어로 이러한 것을 구현하려면 개발자는 30cm, 60cm, 120cm 높이의 점프에 대해 각각의 메소드를 작성해야 한다.