객체지향개발(Object Oriented Programming)

객체 지향임을 밝히는 필수적인 특성으로 나열되는 것은
추상화(abstraction), 캡슐화, 상속성(inheritance), 다형성(polymorphism),동적바인딩(dynamic binding)이 있다.

:실 세계에 존재하는 객체를 소프트웨어로 변환시키는 과정에서 실 객체가 소프트웨어의 객체로 자연스럽게 변환되는 것을 허용한다. 따라서 객체 지향 기법에서는 실세계의 모델링이 용이하며 전체 작업을 객체 위주의 작은 작업 단위로 나누기가 쉽다. 객체 위주로 분리되는 소프트웨어 단위는 이해하고 구현하기가 용이하며 재사용 가능하다.
“객체”와 “클래스”의 구분, 다형 개념의 의미, 추상 클래스의 의미 등이 그 용어를 사용하는 사람마다 다르게 해석되곤 한다. 따라서 객체 지향과 관련된 용어를 이해하고자 할 때 문맥에 따라서 신중을 기해 해석해야만 한다.

***추상화같은 것을 하는 소프트웨어 개발의 본질적인 의미**
소프트웨어를 개발하는 작업의 본질적인 의미는 “simulation”이다. 즉 실세계의 정보나 상황 중에서 주된 관심의 대상이 되는 부분을 컴퓨터 내부로 이식시키는 것이다. 그런데 실세계 상황의 복잡성으로 인하여 실세계의 상황을 바로 컴퓨터 내부로 반영할 수는 없으며 “추상화”와 “구체화” 과정을 거처야 한다. 즉 “추상화” 과정을 통하여 실세계의 상황을 간결하고 명확하게 “모델링”하게 되며 “구체화” 과정을 통하여 추상적 모델을 프로그램으로 변환한다. 시스템의 분석은 “추상화” 과정을 의미하며 이 과정에서 실세계의 상황이 정확하게 모델에 명시가 되지 않으면 프로젝트를 성공적으로 수행할 수 없기 때문에 분석의 중요성이 강조되는 것이다.
이는 마치 음악가들의 사회에서 악보가 그들의 사고를 전달하는 표기법으로 사용되는 것과 같다. 음악가들은 악보라는 표기법을 이용하여 전 세계의 누구라도 같은 음악을 연주할 수 있는 것이다. 소프트웨어의 요구 사항 분석 결과도 마찬가지로 특정한 표기법을 통해서 표현되며 그 결과는 그 표기법에 익숙한 프로그래머들에 이해될 수 있어야 한다.

은닉화
: 클래스를 접근하고자 하는 개발자나 사용자로부터 클래스를 보호나 숨김 시키는 것.
사용자에게는 잘못된 데이터 입력을 방지하고 개발자로부터 필요 없는 정보를 숨겨준다.
"정보은닉"을 표현하는 또 다른 용어는 "블랙박스접근방식"(black box approach)이다.

객체의 부속품
분할정복(divide and conquer)
정보은닉의 원리와 마찬가지로 소프트웨어 공학자들이 오랜 기간 추구해 온 소프트웨어 개발방식.
객체들이 갖는 공통 특징은 대부분 한 객체가 다른 부속 객체들의 조합에 의해 구성된다는 것.

ex) 사람 객체는 머리,몸통,팔,다리 객체들이 모여 만들어지며, 자동차 객체는 엔진,샤시, 바퀴 객체들이 모여져서 이루어짐.
-객체와 부속 객체와의 관계는 has_a 관계로 표시됨.

예를들어, "자동차는 엔진을 구성 요소로 갖는다"라는 문장은 "a car has an engine" 으로 번역되기 때문에 has_a 관계라 한다. 그런데 "엔진은 자동차의 부품이다" "a engine is a part of a car" 는 이 관계는 Part_of 관계.
이러한 관계를 명시하는 용어가 객체 지향 사회에서 "집합화"(aggregation) 라는 이름으로 사용된다.
"집합화"는 "일반화" 경우와 마찬가지로 계층적 트리(hierarchical tree)로 표현되는데 "사람" 구조를 계층적으로 표현한 예이다.

has_a 관계는 다른 말로 "and관계"로 지칭되기도 한다. 그 이유는 "사람은 머리and몸통and팔and다리로 구성된다" 라는 방식으로 기술되기 때문.
시스템을 분석하여 얻게 되는 모델에 포함되는 중요 정보의 대부분은 is_a 관계와 has_a관계들이다.

캡슐화
: 데이터와 함수등 객체와 관련된 것들을 하나로 묶는 것을 말한다. 흔히 정보 은닉과 함께 연관지어 사용되는 개념으로 외부에서 알 필요가 없는 데이터와 행위는 보이지 않게 한다.
객체지향에서 캡슐화라는 개념은 클래스 내부에 여러 속성과 여러 오퍼레이션을 함께 묶음을 의미한다.
그리고 캡슐화는 클래스 내부의 속성이나 오퍼레이션을 외부에 노출하지 않고 보호하는 것을 의미한다.
이렇게 캡슐화는 묶는 것과 보호하는 것을 생각할 수 있다. 좀 더 상세하게 생각해 보면, 여러 속성과 여러 오퍼레이션을 함께 묶어 클래스로 취급하는 것과 클래스 내부를 외부에서 접근하지 못하도록 보호하는 것이 바로 캡슐화이다.

*캡슐화 하는 이유?
묶음으로 인해 프로그램을 바라보는 단위가 커진다. 이전의 프로그래밍 언어인 C언어는 프로그램을 함수 단위로 구조화할 수 있으나, 프로그램 소스가 커지면 이해하기 어렵고 관리가 힘들어 질 수 있었다.
그러나 객체지향 프로그램에서는 프로그램 소스를 클래스 단위로 바라보게 됨으로써 좀더 복잡하고 커다란 소스코드도 쉽게 이해하게 되었다.
왜냐하면 클래스 내부에 여러 함수를 내포할 수 있기 때문에 프로그램 소스 코드를 바라보는 단위가 커졌으며, 그로 인해 프로그램 관리가 좀 더 수월해진 것이다.

두번째, 내부를 숨김으로써 내부를 좀더 자유롭게 변경할 수 있게 되었다.
이전의 함수 중심적인 구조적 프로그래밍 언어에서는 프로그램 내부에서 데이터가 어디서 어떻게 변경되는지 파악하기 어려웠고, 그로 인해 유지 보수가 힘들었기 때문에 자료를 중심으로 함수가 종속되는 구조가 되기도 하였다. 객체 지향에서는 클래스 내부의 데이터를 외부에서 참조하지 못하도록 차단하여 이러한 폐단을 없앨 수 있다. 이렇게 내부의 데이터나 함수를 외부에서 참조하지 못하도록 차단하는 개념을 정보 은닉Information Hiding)이라고 하며 이것이 바로 캡슐화라는 개념이다.

상속
: 클래스의 기능을 확장하는 여러방법 중 하나로 기존 클래스를 수정하지 않으면서도 이미 정의되어 있는 내용을 확장해서 사용할 수 있는 방법을 제공하는 것을 말한다.
예를 들어 자동차라는 클래스를 A가 상속하여 기능을 추가하고 택시 클래스로 정의하는 것을 말한다.

상속은 클래스들의 관계에 있어서 공통적인 속성과 연산을 공유할수 있게 하는 기능.
상속을 하는 근본적인 이유는 기존에 만들어진 클래스를 재사용하기 위한 것.
상위 클래스와 하위 클래스 간에 관계는 벤다이어그램을 이용하여 이해할 수 있다.
객체 지향 과제에서 상속 관계가 차지하는 비중은 매우 막중하다. 그 이유는 상속 관계를 나타내는 트리 구조가 시스템의 골격을 형성하기 때문이다.

기존에 만들어진 클래스에 대하여 한, 두 가지의 자료 구조와 연산이 추가되었다고 하여 새로운 파생 클래스를 시스템에 추가하는 경우가 많은데 이는 지극히 위험하다. 상속 구조를 바람직하게 만들기 위해서는 실 객체가 갖는 본질적 특성을 이해하고 클래스간의 IS_A 관계를 인식하며 차후 변경 및 추가의 가능성을 고려해야만 한다.

상속을 사용하여 좋아지는 것은?

▸ 먼저, 재사용으로 인해 코드가 줄어든다. 하위 클래스에서 속성이나 오퍼레이션을 다시 정의하지 않고 상속받아서 사용함으로써 코드가 줄어든다. 그리고 좀 더 범용성있게 사용할 수 있다.
참고로 하위 클래스는 상위 클래스가 가지고 있는 모든 자료와 메소드를 물려받아 자유롭게 사용할 수 있지만, 또한 자신만의 자료와 메소드를 추가적으로 덧붙임으로써 새로운 형태의 클래스로 발전하게 된다는 것도 잊지 말자.

상속성, 재사용(Inheritance)
- 상위 개념의 특징을 하위 개념이 물려받는 것
- 객체지향의 하이라이트 부분이라고 생각한다. 상속이란 개념이 없으면 객체지향이나 절차지향이나 도진개진
- 간단히 예를 들자면, 자동차라는 부모클래스가 있다.
  기름을 먹거나 달리는 기능을 하는 자동차인데,
  만약 지붕뚜껑이 열리는 특수한 기능을 추가하고 싶다면 기존의 자동차에서 스포차카를 생성한다.
  그러면 스포츠카는 기름도 먹고 달리면서 지붕두껑이 열리는 기능도 갖춘 자동차가 되는 것

다형성(polymorphis)
: 하나의 메소드나 클래스가 있을때 이것들이 다양한 방법으로 동작하는 것을 의미한다.
동일한 조작방법으로 동작시키지만 동작방법은 다른것.

오버로딩, 오버라이딩, 상속 등을 복합적으로 사용함으로써 하나로 여러가지 처리하는 것을 말한다.
예를 들어 오버로딩은 하나의 메서드가 여러가지 기능을 포함하고 있는 것을 다형성이라 한다.

상속을 통해 기능을 확장하거나 변경하는 것을 가능하게 해주고, 같은 클래스 내에 코드의 길이를 줄여주는 것까지 도와주는 개념.
관용적인 개념의 다형성은 같은 모양의 코드가 다른 행위를 하는 것을 나타낸다.

*overloading
가장 이해하기 쉬운 다형성의 예.
하나의 클래스에서 같은 이름의 메소드들을 여러개 가질 수 있게 한다.
단 메소드 인자들은 달라야 한다.
인자들의 타입이나 개수가 다르면 메소드 이름이 같더라도 어떤 메소드를 호출할지 컴파일러가 알 수 있기 때문이다.

참고사이트
https://opentutorials.org/module/516/6127

추상화 obstraction
: 인터페이스와 구현을 분리하는 것을 말한다. 추상화를 통해 객체가 가진 특성 중 필수 속성만으로 객체를 묘사하고 유사성만을 표현하며 세부적인 상세 사항은 각 객체에 따라 다르게 구현되도록 할 수 있다.
간단히 말해서, 어떤 객체를 표현함에 있어서 그 모든 것을 다 표현 하는 게 아니라 일정 부분 특징만을 표현하는 것이다.
불필요한 부분을 제거함으로써 원하는 부분에 집중할 수 있게 만든다.
객체를 표현하는 데 추상화를 잘 하는 능력은 개발자의 필수 능력이다.

자료 추상화를 통하여 정의되는 자료형을 추상 자료형(abstract data type)이라 한다.
구현시에 자료 캡슐화(data encapsulation)를 통하여 이루어진다.

- 공통의 속성이나 기능을 묶어 이름을 붙이는 것
- 객체 지향적 관점에서 클래스를 정의하는 것을 바로 추상화라고 정의 내릴 수 있겠다.
- 좀 더 살펴보면 물고기, 사자, 토끼, 뱀이 있을 때 우리는 이것들을 각각의 객체라 하며 이 객체들을 하나로 묶으려 할 때,
  만약 동물 또는 생물이라는 어떤 추상적인 객체로 크게 정의한다고 하자. 이때 동물 또는 생물이라고 묶는 것을 추상화라고 한다.
- 하지만 무작정 한대 묶으면 되는 것이 아니라
  객체가 맡은 역할을 수행하기 위한 하나의 목적을 한데 묶는다고 생각해야한다. 이것이 많이 들어본 의미로는 은닉화라고 한다.
- 또한 데이터를 절대로 외부에서 직접 접근을 하면 안되고 오로지 함수를 통해서만 접근해야하는데 이를 가능하게 해주는 것이 바로 캡슐화이다.
- 따라서 캡슐화에 성공하면 당연히 은닉화도 자연스럽게 효력이 나타난다.

***그 외 중요한 점**

공급자와 소비자 관점 분리
관점 분리하는 이유는 명세(specification)와 구현(implementation)의 분리를 통한 정보 은닉(information hiding)을 달성하기 위함이다.
같은 소프트웨어 부품에 대하여 공급자의 입장과 소비자의 입장을 나누어 접근할 필요가 있으며 특히 소비자 입장인 경우 최소의 노력만으로 그 소프트웨어 부품에 대하여 이해할 수 있도록 하는 방법이 필요한데 이것이 "명세"이다.
소프트웨어 부품의 "명세"는 그 소프트웨어가 하는 일을 가장 쉽게 이해할 수 있도록 명시한 부분을 의미한다.
한 클래스의 내부 자료 구조와 함수들의 본체가 "구현부"이며 이들은 클래스 공급자의 책임 하에 작성되고 그 클래스의 소비자에게는 숨겨져 있어도 되는 부분이다.
ex)구현부> 아스피린 약의 제조사, 명세부>아스피린의 화학성분 및 제조과정, 질량 등 소비자가 알아야 할 부분