객체지향문법정리 (4월 29일)

객체지향 프로그래밍의 4대 특징

객체지향 프로그래밍의 첫 번째 특징은 추상화 (abstraction)입니다. 추상화라고 하는 것은 객체들의 공통적인 특징(속성과 기능)을 뽑아내는 것입니다. 즉, 우리가 구현하는 객체들이 가진 공통적인 데이터 와 기능을 도출해 내는 것을 의미합니다. 그럼 추상화가 객체지향만의 특징이냐? 그렇지 않습니다. 추상화라고 하는 것은 절차지향 프로그래밍에서도 있어왔습니다. 대표적인 것이 구조체와 같은 사용자 데이터 형입니다. 이것은 데이터를 추상화 해서 하나의 새로운 데이터 유형을 만드는 것입니다. 예를 들면 국어, 영어, 수학 점수를 모아 하나의 데이터 형으로 만든다면 아래와 같이 정의할 겁니다. struct Sungjuk { int kor\_score; int eng\_score; int math\_score; }; Sungjuk이라는 새로운 이름으로 데이터형을 정의 하는 데 그 부속데이터로 kor\_score, eng\_score, math\_score를 두고 있습니다. 성적이라는 데이터를 추상화를 통해 정의한 것입니다. 기능의 추상화도 있습니다. 함수가 바로 그것인데, 잘 쓰는 기능을 이름을 붙여 정의하였으니 기능을 추상화 한 것이죠. 여기서 데이터의 추상화와 기 능의 추상화에서 알 수 있는 하나의 사실이 있습니다. 추상화라고 하는 것에는 항상 새로운 이름이 붙는다는 점입니다. 따라서 프로그래밍 입장에서 추상화라고 하는 것은 이렇게 정리해 볼 수 있습니다. ‘공통의 속성이나 기능을 묶어 이름을 붙이는 것’ 객체지향 프로그래밍에서 추상화는 그럼 무엇일까요? 클래스를 정의하는 과정이 바로 추상화라고 할 수 있습니다.

두 번째 특징은 바로 캡슐화(encapsulation)입니다. 객체지향에서 캡슐화는 데이터 구조'와 `데이터를 다루는 방법'를 결합시켜 묶는 것을 말합니다. 캡슐형태로 된 알약을 떠 올리시면 됩니다. 캡슐형 알약에는 질환을 치료하는 데 필요한 다른 종류의 약이 적정한 비율로 섞여있는 것을 볼 수 있습니다. 해당 질환을 치료하기 위한 목적으로 하나가 아닌 몇 가지 기능의 약을 하나의 캡슐 안에 담은 겁니다. 객체지향에서 캡슐화도 마찬가지 입니다. 특정 객체 가 독립적으로 역할을 제대로 수행하기 위해 필요한 데이터와 기능을 하나로 묶어 관리합니다. 객체가 맡은 역할을 수행하기 위한 하나의 목적을 위해 데이터 와 기능들을 묶는 것이죠. 그런데 캡슐화라는 특징은 다른 의미로도 쓰입니다. 즉, 데이터는 은닉하고 그 데이터를 접근하는 기 능(함수)를 밖으로 노출한다는 의미를 나타낼 때 캡슐화라는 용어를 씁니다. 데이터를 기능이라는 캡슐로 보호한다는 것이죠.

세 번째 특징은 상속성(inheritance)입니다. 상속성은 객체지향의 꽃입니다. 상속이 없으면 객체지향은 절차지향과 큰 차이가 나지 않습니다. 객체지향 문법이 어렵고 복잡해 질 이유도 없죠. 상속이란 특징을 가지면서 객체지향 언어가 절차지향 언어에 비해 문법의 규모 가 2배이상 늘어났습니다. 상속이란 상위개념의 특징을 하위 개념이 물려받는 특징을 말합니다. 예를 들어 ‘남학생’이라는 단어에는 ‘학생’이라는 단어의 특징을 물려받고 있습니다. 개념적 특징이 상속되는 것이죠. ‘학생’이 갖는 특징을 그대로 계승하면서 ‘남학생’만 갖는 특징을 부가 하여 ‘남학생’의 전체 특징이 형성됩니다. 이 개념을 프로그래밍에 도입한 것인데 하나의 클래스가 가지 고 있는 특징(데이터와 함수)들을 그대로 다른 클래스가 물려주고자 할 때 상속성의 특징을 사용합니다.

네 번째 특징은 다형성(polymorphism)인데 객체지향 개념 중 가장 까다롭고 4대 특징 중에 가장 이해하기 어려운 특징입니다. 알았다고 해도 코딩으로 잘 활용이 안 되는 특징이기도 하죠. 이것을 잘 이해하고 활용한다면 객체지향 언어를 제대로 쓰는 것으로 봐도 무방할 정도입니다. 다형성의 의미는 약간 다른 방법으로 일을 하는 함수를 동일한 이름으로 호출해 주는 것을 말합니다. 이해가 잘 안 되죠? 예를 들어 보겠습니다. 홍길동이란 이름의 학생과 김철수라는 학생 이 있다고 가정해 봅시다. 그런데 선생님이 홍길동을 바라 보면서 ‘학생! 칠판 지워’라고 명령을 했다고 합 시다. 홍길동은 나름의 방법으로 칠판을 지울겁니 다. 다시 선생님이 김철수를 바라보면서 ‘학생! 칠판 지워!’라고 명령했을 때 김철수도 자신만의 방법으로 칠판을 지울 겁니다. 그런데 홍길동이 지웠던 방 식과는 다르겠죠? ‘학생! 칠판 지워.’라는 표현은 같지만 칠판을 지우는 행위는 다르게 나타나고 있습니다. 이것이 바로 다형성입니다. ‘학생! 칠판 지워.’라 는 명령이 다양한 결과로 나타나기 때문에 이것을 다형성이라고 하는 것입니다. 프로그래밍 입장에서 다 형성을 해석해 본다면 명령의 해석을 연결된 객체에 의존하는 것으로 정의할 수 있습니다.

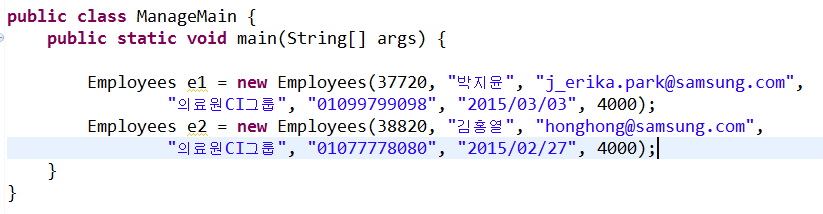
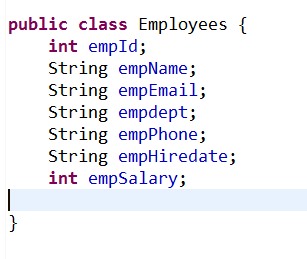
출처 : http://cafe.naver.com/gisdev.cafe?iframe\_url=/ ArticleRead.nhn%3Farticleid=461

-------------------------------------------------------예제 코드-------------------------------------------------------------

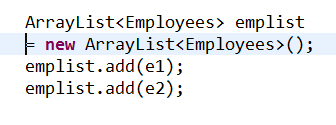
\*추상화가 적용되지 않았을 때의 날 코딩



이미 사원(Employee)이라는 Object의 특성을 알기 위해 중요한 특징들이 간략하게 요약된 형태라 추상화가 어느 정도 된 예제이기는 하지만, emp1,emp2 의 공통적인 특징을 하나로 묶어 재사용성이 좋게 만들어진 예제는 아닙니다. 이 날코딩에서 추상화와 객체의 개념이 적용되었을 때는,

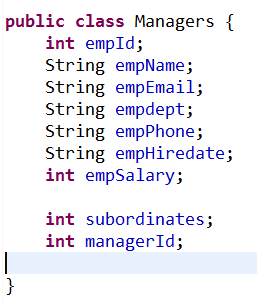
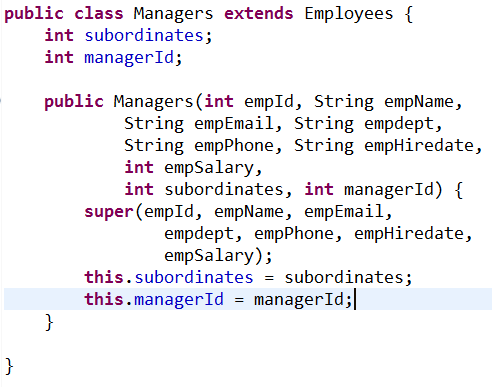


라는 하나의 클래스로 만들어 사원이라는 특성을 정리해두고 재사용할 수 있습니다. 그럼 위에서처럼 emp1~ emp2 라는 변수들을 남발할 필요도 없고 깔끔하죠. 또 사원이라는 이름의 특징으로 묶여진 이 변수들은 하나의 캡슐처럼 사원의 특징을 형성하는데 반복적으로 쓰일 수도 있죠. Java는 이렇게 하나의 클래스도 변수처럼 쓸 수 있습니다. 여러 사원의 정보를 관리할 때 Collections 안에 있는 ArrayList나 Vector를 이용하면 배열보다는 동적으로 자료를 추가하며 데이터들을 관리할 수 있습니다.

 <> 안에 Generics를 지정된 클래스명으로 주면,

하나의 자료구조 안에 넣고 같은 타입들의 정보들이기 때문에 공통된 정보나 필요한 정보만 추출하여 사용할 수 있습니다.

어느덧 신입사원이던 박지윤 김홍열 사원이 매니저가 되어 매니저 클래스가 필요하다고 가정을 해보자면, 이때 그전의 쓰였던 Employee 클래스에 있는 변수들이 대부분 유지되고 끽해야 몇 개의 변수들이 추가될 거에요. 그래서 그렇게 많은 변수들을 또 남발할 것이 아니라 매니저도 결국은 ‘사원’의 한 종류이기 때문에 사원이라는 클래스를 상속받으면 되겠네요?

 그래서 이런 식으로 Employee라는 큰 틀 안에서 공통된 요소는 상속받아 쓰고, 새롭게 추가된 subordinate 부하직원수, 그리고 managerId 라는 새로운 변수들만 추가하여 클래스를 쓰면 됩니다. 그러나 클래스를 extends 확장하여 상속하는 경우의 문제점은 하위클래스가 많아졌을 때 공통된 메서드들을 쓸 때의 문제가 생긴다고 하네요. 그래서 interface를 implement하여 쓰게 됩니다. 고정된 기능들은 같은 이름으로 사용하도록 묶어놓은 것이지요. 클래스를 extends 하면 여러 클래스를 상속받을 수 없지만 interface와 같은 경우 여러 개의 인터페이스를 implements 받아 사용할 수 있기 때문에 유용합니다.