1. 인터페이스 이해하기

객체 지향을 통한 문제 해결 중 각 모듈간의 연결 및 메시지 교환에서 중요한 문제로 부각되는 부분이 바로 인터페이스라고 할 수 있습니다. 많은 사람이 추상 클래스와 인터페이스의 역할에 대한 차이를 정확히 이해하지 못하고 어느 시점에 어떻게 사용되어야 하는지 쉽게 결정하기 어렵기 때문입니다.

그래서 필요한 것은 둘 사이의 차이와 각 언어들에서 어떤식으로 추상 클래스와 인터페이스를 지원하고 있는지를 살펴보고 사용되는 예를 살펴보면 쉽게 둘 사이의 차이를 아해하고 구분할 수 있다면 객체 지향으로 문제를 풀어나가는 과정에서 구현 방향을 선택할 때 하나의 기준으로 사용될 수 있습니다.

추상 클래스에 대한 간단한 설명

인터페이스는 무엇인가?

실 생활에서의 인터페이스의 예

코드로 보기 ( 추상 클래스, 인터페이스 )

1. 객체 간의 메시지 전달

지금까지 설명한 객체지향에 대한 기본 개념을 요약하면 다음과 같다.

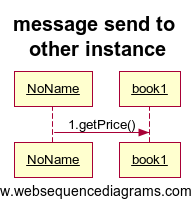
\* 개발할 시스템과 관련이 있으면서 상태와 행위를 제공하는 개별적인 대상을 객체라고 부른다.

\* 유사한 성격의 여러 객체들을 묶어서 클래스라고 정의한다.

\* 상태와 행위를 제공할 수 있는 실체인 객체는 정의된 클래스의 인스턴스로서 생성된다.

요약하면 시스템이 제공할 기능은 클래스의 연산으로 정의되며, 바로 객체가 실제 기능을 제공할 수 있는 실체가 된다. 따라서 시스템이 제공할 기능을 구현하기 위하여 해야 할 다음 작업은 바로 객체가 제공하는 행위를 호출하는 것이다. 객체의 행위는 자신을 생성시킨 클래스의 연산에 의해서 결정된다. 그리고 클래스의 연산은 기존의 구조적 프로그램의 함수와 유사하다. 따라서 객체의 행위가 수행될 수 있도록 하는 것은 기존의 프로그램에서 함수를 호출하는 것과 유사하게 객체가 제공하는 행위(클래스의 연산)를 호출하는 것이다.

그러나 객체지향에서는 “연산을 호출한다” 보다 상위 수준으로 추상화시킨 “메시지를 전송한다”라는 용어를 사용한다. 예를 들어 Book 클래스의 “book1 객체의 getPrice() 연산을 호출한다.”라는 용어 대신에 “book1 객체에게 getPrice() 메시지를 보낸다”라고 표현한다.



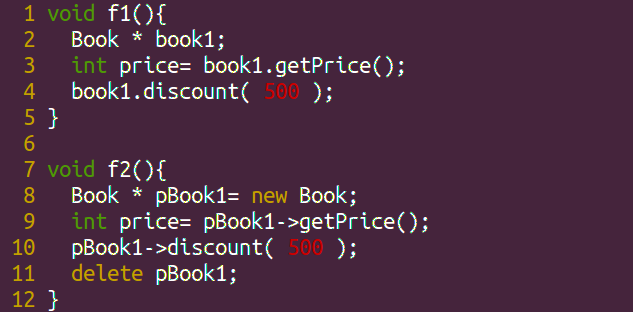
[그림1-11]은 UML의 시퀀스 다이어그램을 이용하여 boo1 객체에게 getPrice() 메시지를 보내는 것을 표현한 것이다. UML의 시퀀스 다이어그램은 객체 간의 메시지 전송을 시간적 순서에 따라 효과적으로 보여주는 다이어그램으로서 클래스 다이어그램과 함께 실무에서 많이 이용되는 대표적인 다이어그램이다. 시퀀스 다이어그램의 상단에는 메시지를 송수신하는 객체가 배치된다. [그림1-11]에서 좌측의 사각형에 객체의 이름을 기술하지 않은 것은 book1 객체에게 메시지를 전송하는 객체의 이름을 표현하지 않은 것이다.

우측 사각형의 book1:Book은 Book 클래스의 객체로서 book1 이름을 갖는다는 것을 나타낸다. 좌측의 이름을 모르는 객체에서 book1 객체의 방향으로 그려진 화살표는 전송되는 메시지를 나타낸다. 즉, 그림[1-11]의 시퀀스 다이어그램은 어떤 객체가 Book 클래스의 book1 객체에게 getPrice() 메시지를 전송하는 상황을 표현한 것이다.

메시지는 임의의 객체에게 전송되는 것이 아니며, 특정한 하나의 객체를 지칭해야 한다. 이는 마치 통신을 할 때 broadcasting 방식으로 불특정 다수에게 메시지를 전달하는 형태가 아니라, 클라이언트/서버의 구조에서 클라이언트가 특정 서버에게 서비스를 요청하는 것과 유사하다.

[그림 1-11]에서도 좌측의 객체는 book1이라는 이름으로 지치오딘 특정 객체에게 메시지를 전송하고 있다. 메시지를 수신한 객체는 메시지에 해당하는 행위(클래스의 연산)를 수행하게 되며, 대부분의 경우에 해당 클래스에 정의된 동일한 이름의 연산이 수행된다. 따라서 위의 예에서는 전송된 getPrice() 메시지에 해당하는 Book 클래스의 getPrice() 연산이 호출되어 수행되는 것이다.

**c++언어에서의 메시지 전송**



book1 객체와 같이 정적으로 생성된 객체에게 메시지를 보낼 때는 book1.getPrice()와 book1.discount( 500 )과 같이 ‘.’ 연산자를 이용한다. 그리고 pBook1과 같이 동적으로 생성된 객체에게는 ‘->’ 연산자를 이용하여 메시지를 전송한다.

**Java 언어에서의 메시지 전송**



main() 메소드에서는 Book 클래스의 객체를 생성하고 이를 book1 변수가 가리킨다. 그리고 book1.getPrice() 문은 book1 변수가 가리키는 객체에게 getPrice() 메시지를 전달하는 것이다. Java 언어에서는 객체에게 메시지를 전달할 때는 항상 참조 변수(book1)와 메소드 (getPrice()) 사이에 ‘.’ 연산자를 이용한다.

1. 실패에 안전한 환경 ( 소스 형상 관리 )
   1. 소스 형상 관리는 왜 필요한가?

우리는 소프트웨어 시스템 및 하드웨어의 펌웨어 소프트웨어를 작성하면 많은 양의 소스 코드 및 프로젝트 관련 자료를 생성합니다. 매 순간 매 시간 동안 많은 양의 데이터가 생성되거나 혹은 변경되고 삭제됩니다. 어제 잘 작동하던 코드가 오늘 소스 코드의 특정 부분을 수정한 이후로 예기치 않은 동작을 한다든지 하는 일이 종 종 일어납니다.

이러한 환경에서 우리가 할 수 있는 일은 매 순간 잘 작동하는 코드를 파일 압축과 같은 형태를 통해서 자료를 모으고 파일 이름에 버전 및 날짜에 대한 정보를 기입함으로써 관리하곤 합니다.

실제로 이러한 방법은 가장 간단하고 빠르게 진행할 수 있는 고전적이면서도 빠른 방법입니다.

소프트웨어 배포에 대해서도 같은 맥락에서 압축 파일을 통한 배포가 사용되기도 합니다.

하지만 아쉽게도 이러한 방법은 큰 실수를 불러오곤 합니다.

예를 들어보자면 기존에 잘 작동하는 소스에 최근 수정한 코드가 포함된 파일이 있는 압축 파일을 덮어쓰기 한다든지, 여러 버전에 대한 압축된 파일이 존재하게 되면서 시간이 흐를 수록 잘 작동하는 버전을 찾고 관리하는데 들어가는 시간이 많아진다는 것입니다.

**\*압축된 여러 버전 파일이 폴더에 존재하는 스크린샷?**

그래서 이런 문제를 해결하기 위한 하나의 방법이 소스 형상 관리를 이용하는 것입니다. 이를 통해서 어제 잘 작동하던 소스 마지막으로 잘 작동하던 소스로 현재 작업 중인 소스를 갱신하거나 현재 작업 중인 소스가 오작동 할 때는 가장 최근 잘 작동하던 소스로 되돌리기 할 수 있습니다.

실패를 하더라도 가장 최근에 잘 작동하던 소스로 되돌릴 수 있다는 말은 곧 현재 지금 작성하고 있는 소스에 대해서 다양한 시도를 해보고 만약 실패하더라도 최근의 가장 잘 작동하는 소스 즉 모습으로 되돌릴 수 있으니 현재 소스 변경에서 얻을 수 있는 오작동, 오류에서 오는 실패에 대한 안정된 환경을 통해서 심적 부담을 덜고 다양한 시도를 통해서 프로젝트를 개발해 나갈 수 있게 됩니다.

* 1. 기본적인 동작 매커니즘

Checkout/

Modify/

Update/

resolve/

Commit/

각 단계에 대한 설명을 간단하게 해줄 것.

* 1. 소스 형상 관리 툴의 종류

Cvs

폴더 단위로 각각 소스의 버전을 관리함.

Svn

원자 단위로 모든 파일 디렉토리에 대한 버전을 관리함.

git

cvs, svn의 checkout/commit 구조가 아닌 checkout/ commit/ affect

* 1. 소스 형상 관리를 통한 개발 프로세스 살펴보기

소스 형상 관리를 이용해서 소스를 공유하고 관리하는 과정은 릴리즈(Release), 분기(Branch), 메인소스(Root)로 크게 나누어집니다.

개발 중인 버전에서 데모를 위한 안정화 버전 작업이 필요할 때에 대한 사례연구.

1. 참고문헌

[1] 클래스 구조의 이해와 설계- 메시지 31p~33p, 한빛미디어, 2004