Interface 설계 – Part 4

Part 3에서 Interface 구현과 OOP의 상속에 대한 차이를 설명하려다가 삼천포로 빠진 경향이 있네요. 아마 배가 고파서 일겁니다.

오늘은 OOP의 상속에 대해 얘기하고, Interface가 OOP 상속의 대안이 될 수 있음을 설명하겠습니다.

모든 면에서 그렇지만, 항상 장점이 있으면 단점이 있고, 변증법은 아니더라도, 새로운 대안이 나오기 마련입니다.

자 그럼 OOP의 가장 기본적은 개념부터 알아봅시다.

OOP (Object Oriented Programming)은 기존 Procedure(Function) 스타일의 프로그래밍 방식의 문제점을 극복하기 위해서 고안된 것이다. 기존 Procedure 방식은 Data 처리에 중점을 두고, 절차적인 처리를 중심으로 프로그램을 수행하여, 복잡한 시스템의 경우, 프로그램 제작이 어렵고, 재활용성이 상당히 떨어지는 경향이 있었습니다.

자세한 OOP의 내용은 <http://www.joinc.co.kr/modules/moniwiki/wiki.php/article/oop> 에 친절하게 한글로 되어 있으니 참고하시기 바랍니다. – class 가 중심임을 강조합니다.

사실 OOP를 처음 접했을 때, 가장 먼저 접한 단어는 “재활용성”이었습니다. 이 단어를 가지고, 추상화, 상속, 다형성 등등의 OOP 특징을 장점으로 내세웠지요.

항상 그렇듯이, 설계는 아는 만큼의 지식을 가지고 실행하는 것이고, 일이 진행되는 도중에 상당히 많은 부분에 대해 재설계가 됩니다. 어쨌든 class 구조가 잘 설계되었다면 사용자나 응용 개발자 모두 편안할 것입니다.

OWL, MFC, Delphi 같은 경우, class 구조가 상당히 바뀌기도 했지만, 상당히 견고한 구조로 구성되어 있어, 많은 개발자들이 그 소스를 보고 배웠을 것입니다.

몇 년이 흐른 후, 어느 정도 OOP에 익숙해 질 때 (상속, 다형성 등을 무리없이 사용/제작할 때) 문득 몇가지 의문이 들었습니다. 정말 내가 class 구조를 잘 설계해서 도움이 되었는가? 이런 구조를 제작하기 위해 너무 많은 비용이 투입되는 것은 아닌가 하는 의문이 말입니다.

OOP 개념 자체가 문제가 아니라, 그 개념을 실현하는 사람의 능력에 따라 class 구조의 견고함이 크게 달라지기 마련입니다.

다시말해, 도구나 방법론이 아무리 좋더라도, 다룰 줄 모르거나, 실행방법을 모른다면 무용지물이고, 애물단지 일 뿐입니다.

뭐 그렇다고 OOP 형태로 개발하지 말자는 얘기가 아니라, 실제로 OOP 를 이용하여, OOP의 장점을 극대화 시키려면 상당한 노력과 경험이 필요하다는 것입니다.

OOP의 상속은

1. Class 들의 공통점을 추상화하여 base class를 제작하고
2. 상속한 sub-class는 자신만의 고유 특성을 추가한다. (확장)
3. base class <- sub class 형태가 되고, base의 signature를 sub class에서 재정의를 하여 다형성을 제공한다.

보통의 WEB 개발장의 경우, 다뤄야 할 객체를 class화 하는 일도 하지 않는 경우가 많은데(DataSet 이용), 추상화 단계를 거쳐 class를 구조화할 필요도, 여유도 없었을 겁니다. (class 구조로 된 .NET, javascript도 사용하지만)

앞서 말씀드렸듯이 class를 구조화 한다는 일이 쉽지 않고, 노력/경험이 많이 필요하기는 합니다.

간단하게나마, Site에서 Customizing 할 때, 기존 class를 수정하지 마시고, 상속을 이용하여 작업해 보십시요. 상당한 도움이 될 것입니다. – “사실 TABLE이 변경되면 Class를 수정하는 것이 옳지 않느냐?” 라는 의문이 들겠지만, 그건 다른 방법으로 해결해야 할 수 있습니다. – 이렇게 시스템이 확장이 되어 갈 수 있겠죠…

어? Class 상속(inheritance) 이나 interface 구현이나 재정의나 구현이냐의 용어 차이와 base class의 구현된 코드를 재활용할 수 있다는 것만 빼고는 똑같지 않습니까?

어쨌든 상속이던 구현이든 재정의를 한다는 것과, 확장을 한다는 것은 동일합니다.

가장 큰 차이점은 class 구조를 설계할 때의 추상화 단계에서 달라집니다.

1. 단일 상속과 다중 상속  
   앞서 말했듯이 C#, java 같은 OOP 언어는 다중 상속을 지원하지 않고, 단일 상속만을 지원합니다. C++는 다중상속을 지원하지만, 여러모로, 다중 상속의 편리한 점보다는 상속 구조의 견고함을 깨뜨릴 위험성이 많다는 것이 중론입니다. 어쨌든 단일 상속만을 지원하는 class 상속과 다중 상속(구현이 더 맞는 표현)이 가능한 interface 구현은 더 자유로울 수 있습니다.
2. 추상화 단계에서의 문제점  
   BPA 프로세스 맵에서 사용되는 Node, Link 들을 추상화 한다고 가정해 보면, 아주 복잡한 상속 체계가 나올 것이고, 어떤 상속체계를 구성해야 할지, 여러 usecase 별로 달라질 것입니다.  
   분류학이라는 학문에서 보듯이, 분류체계 (상속체계와 유사) 라는 것이 관점에 따라 상당히 많이 나올 수 있고, 이를 통일된 하나의 체계에서 설명한다는 것이 너무나 어렵다는 것입니다.

결론적으로 Interface의 장점은

1. 다중 상속이 가능하고, 추상화 단계에서 통일된 체계에서 벗어나는 관점을 보완할 수 있다는 점
2. 극한의 추상화로, interface 구현체는 자신만의 코드를 가지므로, base class의 내부 변화와는 상관없이, 독립적으로 코드 구현이 가능하고, 상속 체계상의 변화에 영향을 덜 받는다.

Interface로 하지 말아야 할 것.

* + - 1. 복잡한 속성/메소드를 한 interface에 정의하지 말 것.  
         추상화가 덜 된 것으로, interface를 쪼개던가, 더 추상화를 수행 할 것  
         현재 RealBPA에는 Entity별로 interface를 모두 정의했는데, 오바라고 봐진다.
      2. Interface의 signature를 자주 변경하면 안된다. 특히 공개 이후에는 변경이 불가능하다고 봐야 한다.  
         interface signature가 변경된다는 것은 계약관계가 변경된 것이므로, 모든 구현체가 변경되어야만 한다. 이를 방지하기 위해 새로운 버전의 interface를 추가하는 방식을 사용해야 한다.  
         언듯 보기에 base class도 비슷하지만, base class는 자신만의 고유 signature와 상속 불가능한 정보도 상당히 많기 때문에, 외부가 아닌 상속 체계에서 문제를 일으킬 소지가 많다.  
           
         또하나의 큰 차이점은 interface의 signature 변경은 compile시에 찾을 수 있지만, base class의 구현부분의 변경은 실행해 봐야만 찾을 수 있을때가 많다.

이만 Interface 관련 내용을 끝마치겠습니다.