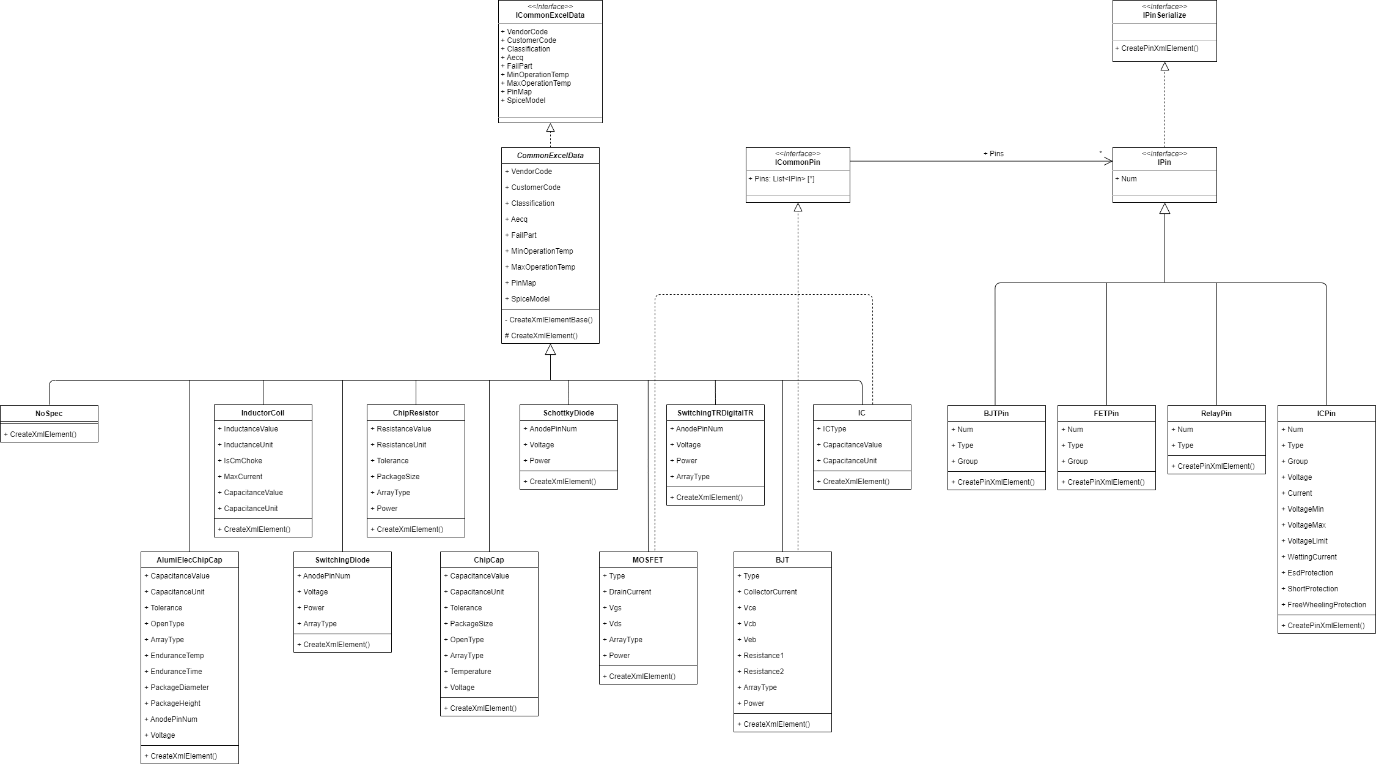
**현장실습 주간보고서[6주차]**

**21.02.08 ~ 21.02.10**

**아주대 미디어학부 황지훈**

**펜타큐브-개발**

**1. UML Class Diagram**

****

XML 파일로 저장하는 과정에 필요한 주요 Class를 대상으로 UML Class Diagram을 작성했다. 이 과정의 선행과정(엑셀파일에서 부품 정보를 의미하는 class로 로드)에 필요한 class들(DefaultExcelParser, ParsedExcelData, XMLCreator, MainClass 등)은 Class Diagram에서 생략했다. 또한 class diagram에 포함된 class들의 property, method들(엑셀 column의 row를 의미하는 property, 엑셀 데이터를 class로 parsing하는 메소드 등)도 일부 생략했다.

**2. C# 부품 도메인 클래스 -> XML 파일로 저장**

XML 파일생성 역할을 담당할 class(XMLCreator)를 생성했다. XMLCreator class에 SaveXML() 메소드를 정의하여 XML 파일을 생성하도록 했다. 생성해야 할 XML파일의 스키마는 Serializable을 사용할 수 없는 구조이다. 따라서 XDocument, XElement, XAttribute 라이브러리를 사용하여 XML파일을 구성했다.

부품 정보를 가장 잘 알고 있는 부품 도메인 클래스에서 XML의 Part Element를 생성하도록 구현했다. 따라서 각 부품 클래스에 XElement를 반환하는 CreateXmlElement() 메소드를 정의하고 SaveXML() 메소드에서 각 부품 class의 CreateXmlElement() 메소드를 호출하여 Part Element를 반환 받도록 했다. 또한 핀에 대해 가장 잘 알고 있는 4개의 Pin 도메인 클래스가 XML의 Pin Element를 생성하도록 구현했다.

ICommonPin 인터페이스를 정의하고 Pin Element를 갖는 4개의 Part 도메인 클래스가 이를 구현하도록 했다. ICommonPin에 대한 4개의 Part 도메인 클래스에 Polymorphism 패턴을 적용했다. 따라서 ICommonPin을 구현한 Part 클래스가 ICommonPin로 캐스팅 가능하다면 Pin을 갖는 Part를 의미하게 되고, 이 경우에 대해서 XML의 Pin Element를 생성할 수 있도록 했다.

**3. partial, sealed 키워드**

클래스 정의를 분할하는 것이 바람직한 상황에서 partial 키워드를 사용한다. partial 키워드는 클래스, 구조체 또는 인터페이스의 다른 부분을 네임스페이스에서 정의할 수 있음을 나타낸다. 예를 들어 클래스의 property 정의 부분과 method 정의부분을 나누어 정의할 수 있다. Sealed 키워드로 선언된 class는 상속이 불가능한 class임을 명시한다. Partial class에 sealed또는 abstract는 분할된 class들 중 하나에만 명시하여도 무방하다.

**4. virtual, abstract class**

virtual 키워드는 method, property 선언을 한정하는데 사용된다. Child class에서 필요에 따라 override할 수 있지만 필수사항은 아니다. 하지만 abstract 키워드가 사용된 class를 상속하는 child class(concrete class)는 반드시 abstract method를 override해야 한다.

**5. 코드에서 directory path 지정하기**

Directory 경로를 표현하는 방식은 운영체제 등에 따라 차이가 있다. 예를 들어 Window 시스템에서 directory 구분을 ‘\’로 하지만, UNIX 시스템에서는 ‘/’를 사용한다. 따라서 절대적인 경로 위치를 string으로 명시하기보다, Path의 Combine() 메소드를 사용해서 directory 경로를 지정하는 것이 옳다.

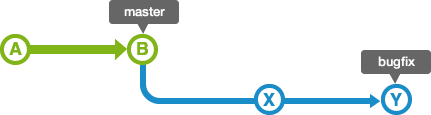
만약 Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "ouput")과 같이 경로를 표현하면 실행파일이 존재하는 위치 아래에 output이라는 directory를 의미하는 경로가 된다. Combine() 메소드의 인자는 여러 개가 될 수 있으며 순서대로 join하여 경로가 만들어진다.

**6. C# params 키워드**

params 키워드를 사용하면 가변 개수의 인수(1차원 배열)를 사용하는 메소드의 매개 변수를 지정할 수 있다. 매개변수 배열은 1차원 배열만 가능하다. 메소드의 매개변수 선언에서 params 키워드로 선언된 매개변수는 반드시 하나만 존재할 수 있고, 이것은 반드시 가장 뒤에 선언되어야 한다. 예를 들어 void func(params object[] arr)과 같이 정의된 메소드를 호출할 때, func(1, 'a', "test") 또는 func(1, 2, 3, 5, 9.3, 2) 등과 같이 호출이 가능하다.

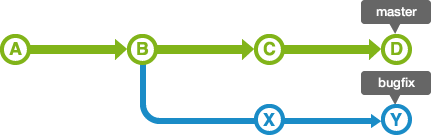
**7. git merge**

**1) fast-forward-merge (branch 이동)**

****

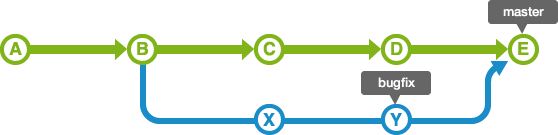
위 사진은 master(main) branch에서 bugfix branch를 생성하여 작업한 예시 상황이다. 위와 같은 상황에서 bugfix branch를 master branch로 merge할 때, master에서 branch를 생성한 뒤에 commit을 하지 않았다면 쉽게 병합할 수 있다. bugfix branch의 이력은 master branch의 이력을 모두 포함하고 있기 때문에, master branch는 단순히 이동(rebase 또는 merge)하기만 해도 bugfix branch의 내용을 적용할 수 있다. 이 같은 병합은 'fast-forward-merge'라고 한다.

**2) non-fast-forward-merge (git merge)**

****

위와 달리, bugfix branch를 분기한 이후에 master branch에서도 변경 이력이 존재(master branch에서 커밋 진행)하는 경우를 생각해보자. 이 경우에는 master branch 내의 변경 내용과 bugfix branch 내의 변경 내용을 하나로 통합할 필요가 있다.

병합하는 과정은 간단하다. 우선 병합할 branch로 이동 후, git merge <병합될 branch>를 입력하며 merge를 진행한다.



위 사진은 bugfix branch의 이력을 master branch로 병합한 그림이다. 위와 같은 병합을 하기 위해, git checkout master, git merge bugfix를 순서대로 입력하면 된다.

**8. git fetch, pull**

Pull의 경우 원격 저장소로부터 필요한 파일을 다운하고 merge까지 진행한다. Local branch와, 원격 저장소 origin/master 가 ‘같은 위치’(가장 최근 커밋)를 가리킨다. Fetch는 원격 저장소로부터 필요한 파일을 다운하지만 merge는 진행하지 않는다. Local branch는 원래 가지고 있던 local 저장소의 최근 커밋 위치를 가리키고, 원격 저장소 origin/master는 가져온 최신 커밋을 가리킨다.

Fetch를 사용하면 원래 내용(원격 저장소의 내용)과 local에서 작업하여 변경된 내용의 차이를 알 수 있다.(git diff HEAD origin/master) 또한 origin/master로부터 커밋이 얼마나 진행되었는지 알 수 있다. 따라서 fetch를 사용하면 원격 저장소와의 세부 내용 비교 및 확인할 수 있기 때문에(Merge를 하지 않기 때문에) 신중하게 접근해야 할 경우 사용되며 git merge origin/master를 하면 git pull 상태와 같아진다.

**9. Reference**

- <https://stackoverflow.com/questions/6701292/git-fast-forward-vs-no-fast-forward-merge>

- https://backlog.com/git-tutorial/kr/stepup/stepup1\_4.html