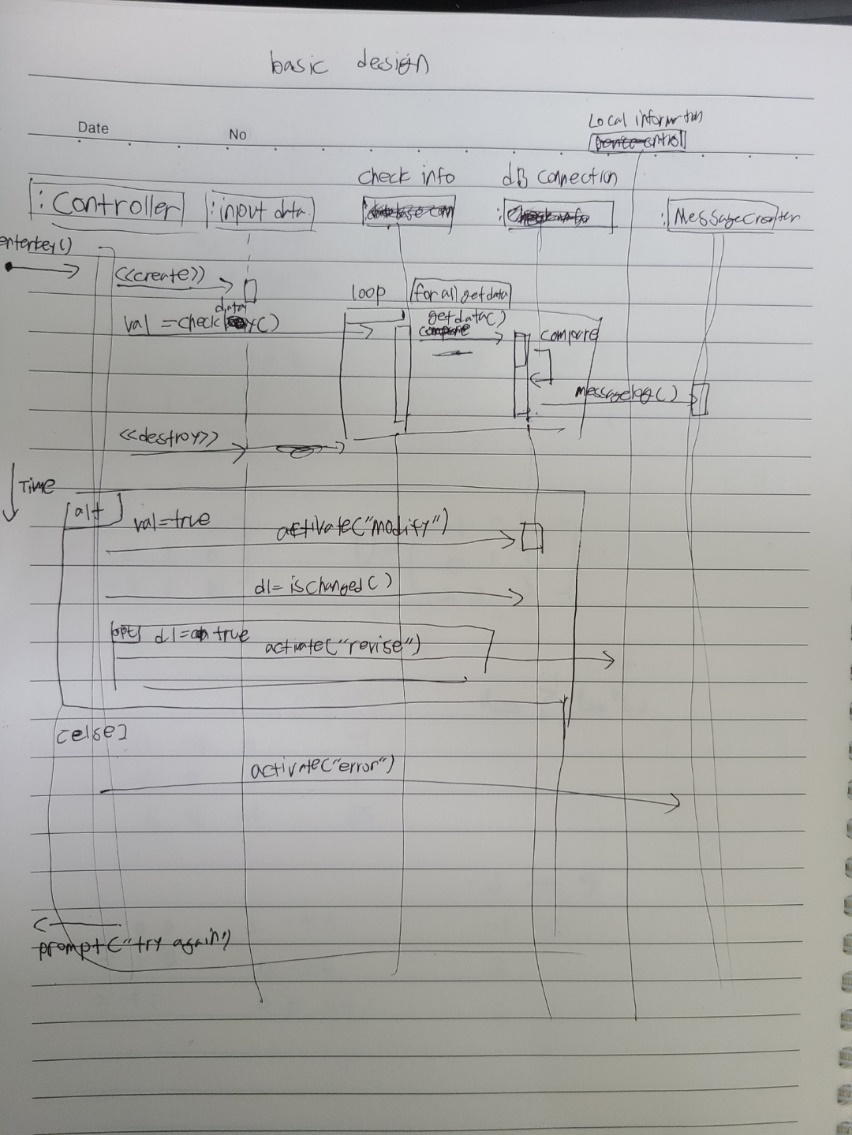
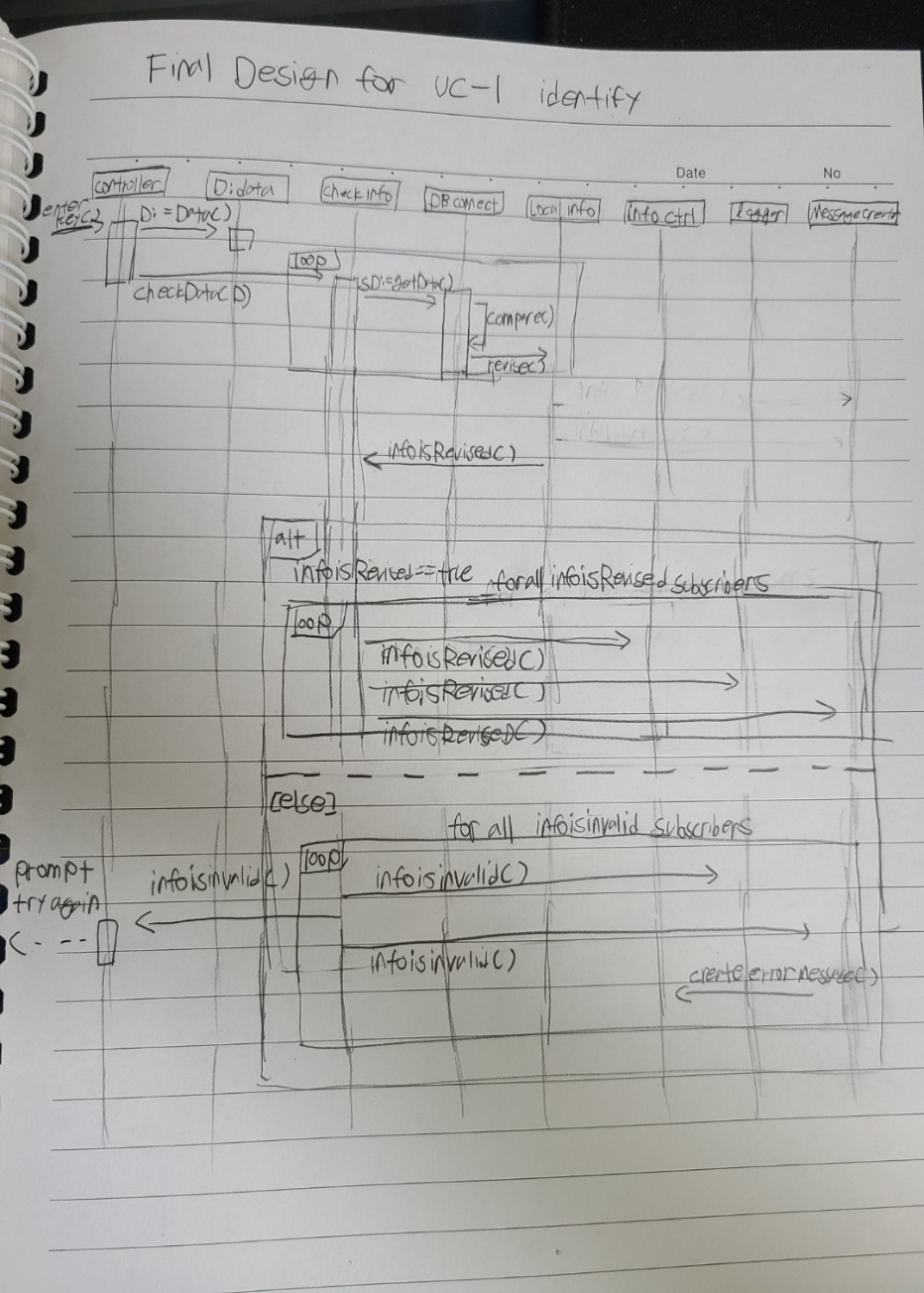
Object sequence use case 1: identify 최초구상

Object Sequence 최초 설계에서는 domain model을 본따 시간의 흐름에 따른 data의 flow랑 조건에따른 정보의 이동을 간략화 하였습니다.

이 때, domain model에서 구현되지 않았던 object인 local information을 설정함으로써 프로그램 내부의 정보를 기록하는 저장소를 만들어 데이터의 흐름을 보다 원활하게 만들었고, data를 check함으로써

유효하지 않은 data라면, 정보 입력을 취소하고 재입력 시도하는 기능을 추가했습니다.

이 과정에서 data의 흐름에 따른 knowing과 doing의 구분이 명확하지 않고 과정이 번잡하기에 Pub-Sub pattern 방식을 적용하여 새로운 기능이나 옵션이 추가되어도 부분적인 수정만으로 충분히 이루어낼 수 있고, 데이터의 흐름과 관리를 더욱 수월하게 하기로 생각했습니다. 또한 기존에 정의 되지않았던 정보의 변화에 대한 변경점의 도당점인 info ctrl과 기록을 남기는 logger 그리고 몇몇 object의 이름을 변경하여 최종적인 object sequence diagram을 아래와 같이 재구현했습니다.

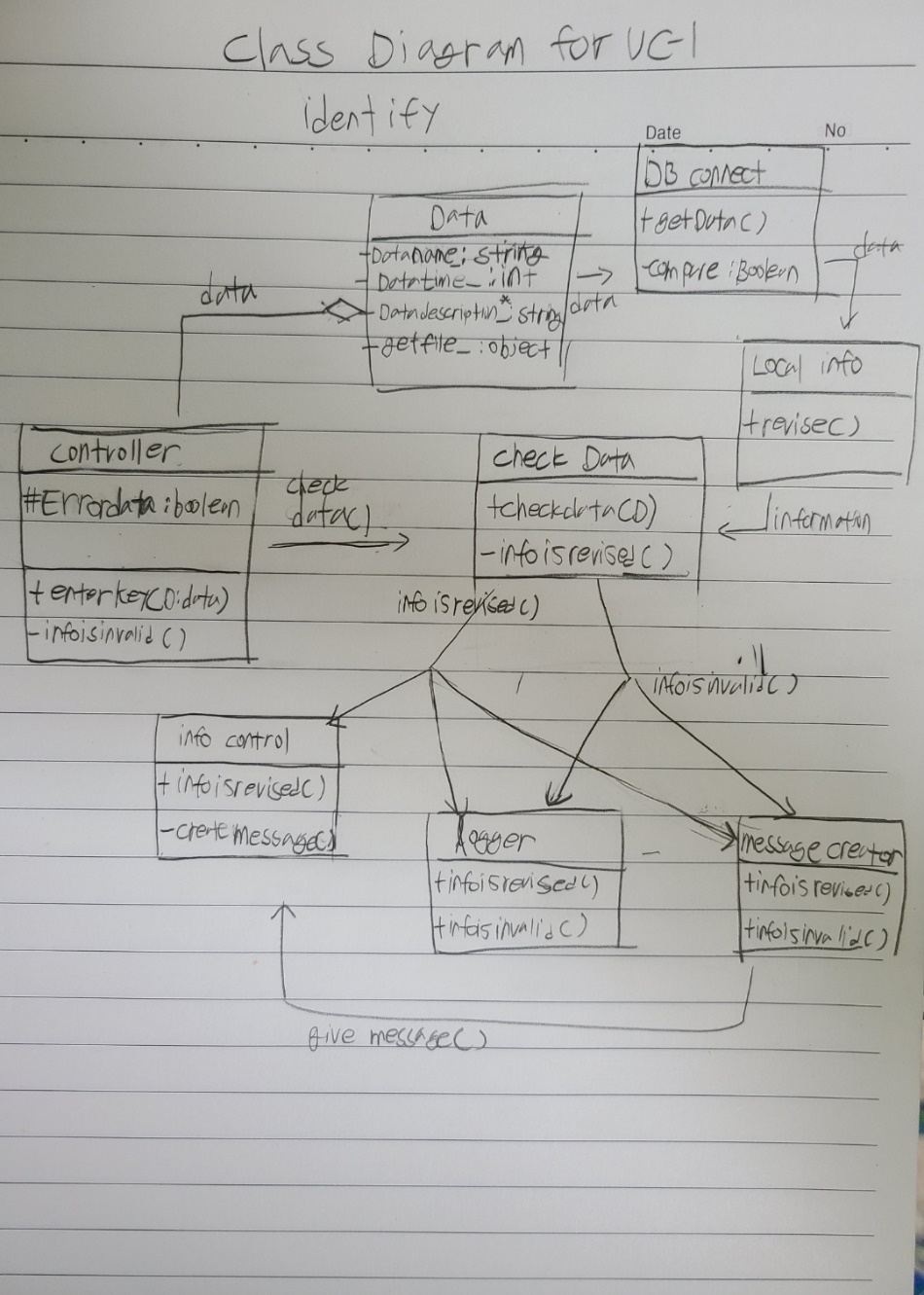
Object Sequence Diagram 최종버전

Info ctrl(정보관리를 위한 object), logger(정보 입력에 대한 log 저장), message creater(사용자에게 보여지는 메시지를 만드는 장치)를 기본적인 구독자로 설정함으로써 정보의 입력에 따른 결과가 변경되었을 경우에 대한 사항들을 구독함으로써 그에 따른 처리를 수월하게 만들었습니다. 그리고 만약 데이터가 유효하지않을 경우 에러메세지와 함께 controller에게 다시 시도하라는 명령을 처리하게 하는 기능을 보다 획일화시켰습니다.

이외에도 여러가지 design principle을 준수하여, cohesion을 높이고 coupling을 최대한 낮추어 적당한 밸런스를 이루어 시스템의 의존성을 낮춤으로써 위험부담을 줄이고, 가시성을 높여 더욱 효율적인 object sequence diagram을 구현했습니다.

마지막으로 class diagram은 앞서만든 RAD의 결과물인 domain model과 object sequence diagram을 바탕으로 Solid의 dependency inversion 형식을 따서 만들었습니다.

Class diagram 최종버전



일단, Data에는 data의 기본바탕이 되는 형식들을 작성하여 data의 기본구조를 나타내었고, controller가 모든 것을 결정하기보다는 각자의 object들이 명령을 수행할 수 있도록 분산구조를 만들어놓아 coupling을 낮췄습니다. 또한 직접적인 Data와 Data의 변경여부를 알려주는 구조를 분산하여(Dependency inversion 방식 참조) 이후 수정에도 쉽게 해놓아 유지보수에도 강한 면모를 보이는 시스템으로 구조를 짜보았습니다. 기본적으로 Object sequence diagram과 domain model 설계과정에서 많은 고민을 거쳐왔기에 앞서만든 object sequence diagram 처럼 많은 버전의 수정은 없었지만, SOLID 원칙의 Dependency Inversion Principle을 적용시켜 상위 모델을 하위 모델에 의존하지 않도록 설계함과 Pub-Sub 형태로 diagram을 구축함으로써 효율적인 설계를 만들었습니다.