1. 1 달의 시간 동안 느낀점과 간략한 소감을 작성
   : 업을 바꾸기로 마음을 먹은 후로, 그 간격을 최소화 하기 위해 많은 준비를 하지 못했습니다. 이 분야에 대한 관심도는 매우 높았고, 적성에도 맞을 것이란 자신감이 높았고, 그래서 주저없이 시작했습니다.
   때문에 기본 지식은 전무했으며, 수업에서 마주하게되는 것은 모두 처음 접하는 것이었습니다.
   가정이 있다보니 집에서 복습은 매우 한정적이었고, 학습내용을 숙지하는데 많은 어려움이 있습니다.

   그런데 돌이켜보면, 학창시절에 배우는 원론적이고 문법적인 내용들은 취업 후 사회에서 필요로 하는 스킬과는 거리감이 있었습니다. (더욱 발전하기 위해서는 학교 수업 내용이 필요하겠지만)
   제가 아직 이 분야의 현업을 경험하지는 못했지만, 이 수업을 통해서 협업과 코드 디자인 방식, 그리고 프로젝트를 대하는 자세 등을 배워서, 수강생들이 취업 이후에 마주하게 될 상황에 대한 대응 능력을 기를 수 있다 생각되어집니다.

   2 번의 팀 프로젝트를 통하여, 협업하는 과정을 체득할 수 있었고, 얕게나마 우리의 문제점을 알고 수정해 나갈 수 있는 배움의 시간이었습니다.
   저는 여전히 코드의 기본적인 지식은 매우 빈약하지만, 강사님의 가르침대로 코드는 결국 누구나가 알게 되는 것이고, 그 이상이 필요하다는 것에 저도 공감합니다.
   짬짬히 집에서 자습하면서 강의 내용을 하나씩 더 이해할 때마다, 제가 성장한다는 느낌을 받고 있어서 공부하는 동기부여가 되고 있습니다. (학창시절에도 무언가를 배우고 알게되는 것에 재미를 느꼈습니다.)

   최근에 혼선이 되는 것은 c 언어에서 c++ 로 넘어가면서 코드 표기법이 일부 변경이 되었고,
   파이썬으로 넘어가면서 코드 표기법이 또 일부 변경되었습니다. 우리 수업의 최종은 러스트(c, c++ 모두 사용하겠지만) 까지인데, 각 언어를 어디까지 숙련을 해야할지 고민입니다. 지금의 숙련보다는 더 실력을 키워야 하는 것은 분명한데, 저에게 허락된 시간은 한정적이므로, 각 언어에 대한 몰입 정도를 정하기가 쉽지 않습니다.
   강사님께서 문법에 대한 강조가 깊지 않은 것이, 세부사항에 빠지지 않기 위함인데, 만약 지나가는(개념 정도만 익히고자 했다면) 언어에 대해 제가 힘을 빼는 것은 아닌지 모르겠습니다.

   그러나, 이와 같은 고민은 하기에는 아직 코딩 문법적 실력이 바닥이라서 현재로는 지속적으로 공부할 예정이며,
   앞으로 개인 프로젝트와 최종 프로젝트가 시작할텐데, 저의 역량을 빠르게 키워서 프로젝트를 함께 해 나가 싶습니다. 가족들의 양해를 얻어서 에너지를 더 쏟아야 할 것 같습니다.

2. Domain Driven Design 이점들을 기술
   : 한 가지 아젠다를 이루기 위한 Domain 들을 작성을 하여, 분업 수행, 반복 작업 방지 등을 통하여 생산성을 극대화 할 수 있습니다.
   코드 가독성이 좋기 때문에, 유지 보수가 쉽고 제 3자의 이해와 접근이 쉬워집니다.

3. TDD를 통해 얻을 수 있는 이점

: 분업화 하여 진행한 코드의 정상 작동 여부를 확인하여, 각 팀원들의 업무결과를 합쳤을 때
문제 발생 원인을 최소화 할 수 있습니다.
테스트를 하고자 하는 이유를 전달할 수 있기 때문에, 테스트 내용을 통하여 코드의 목표를 쉽게 이해할 수 있습니다.

4. DDD와 TDD의 관계 대해 기술
: 도메인 단위의 업무를 진행하고, 그 작업물의 완성 여부를 확인하 위해 도메인 단위로 테스트를 합니다. 테스트는 기능 관점이 아니라, 해당 도메인의 목표가 이루어지는지를 확인합니다.

5. 팀원들과 협업을 할 때 중요하다 생각하는 요소들을 작성
: 소통 -> 각자 맡은 분야에서 작성한 이후, 결과물을 합치는 과정에서 함수명이나 변수명이 달라서, 재작업을 하는 경우 발생할 수 있습니다. 지속적으로 소통을 하여 서로의 연결 고리가 될 수 있는 부분은 공유하여 작업해야합니다.

백로그-> 백로그를 통하여 팀원의 진행상황과 앞으로 팀에 필요한 부분을 파악할 수 있으며, 이를 통해 중복과 불필요한 작업을 미연에 방지할 수 있습니다. 또한 내가 필요로 하는 부분을 팀원이 미리 작업을 하였다면, 해당 건을 참조하여 생산성을 높일 수 있습니다.

소속감-> 팀원으로서 프로젝트를 함께 만들어 간다는 소속감을 유지시켜서, 힘들거나 난관을 마주하더라도 팀원들과 의견을 나누고 서로 도움을 주어 문제를 해결해 나가야 합니다.

6. 함수 포인터 테이블을 사용하는 이유(IoC 관점)
: 함수 포인터 테이블을 통하여 각 기능이 작동하는 코드들을 호출합니다. 이를 통해 작업을 구성하는 것과 작업을 수행하는 것을 분리하게되며, 기능이 추가 삭제하는 업무가 편리해집니다. 기능간의 간섭이 없으므로 분업이 가능하며 생산성도 높아집니다.

7. virtual method를 사용하는 이유
: '이러한 함수를 사용할 것이다' 라고 미리 정의를 합니다.

8. 협력형 과제를 진행 중 어려운 문제를 마주하면, 어떻게 대응 할 것인지 기술
: 문제의 원인을 파악하고 관련된 팀원과 소통을 합니다. 문제해결이 이루어지지 않는다면 도움을 얻기 위하여 다른 팀원과 회의를 합니다. 문제를 해결하기 위해서는 목표가 무엇인지 되짚어보고, 발생한 문제가 목표의 어떤 점을 방해하거나 막고 있는지를 생각합니다.
문제를 제거하거나, 다른 우회 방법을 통해 목표를 해결할 수 있는지를 통해 다각도로 분석하고 대안을 생각합니다.

9. Backlog 이점
: 팀원의 진행상황을 실시간으로 파악이 가능하여, 현재 작업의 이전과 이후를 빠르게 이해할 수 있습니다.
문제가 발생하였을 때는 해당 작업구간을 찾아 대응하기가 수월합니다.
작업의 목표를 명확히 할수 있으며, 이는 작업 방향을 상실하지 않고 올바르게 유지하는데 도움이 됩니다.

10. 분업의 결과물들이 원활하게 결합하지 못하 폐기처분는 경우의 근본적인 이유를 기술
: 팀의 작업 내용들을 파악하지 못하여 작업의 결과물이 팀이 나아가고자 하는 목표와 다를 수 있으며, 또는 이미 작업한 내용과 중복이 될수 있습니다.
소통의 부족으로 작업의 우선순위를 인지하지 못하고, 프로젝트가 진행되면서 방향성이 변경되었을 때 폐기도 합니다.
분업을 진행하되, 팀 프로젝트가 나아가는 방향은 일치되어야 합니다.

11. 기능 단위로 Backlog 를 작성하는 경우 발생하는 문제를 기술
: Backlog 의 목표를 알기 어려워지며, 누가 어떤 목적을 갖고 이 기능을 사용하는지를 파악할 수가 없습니다.

21. 실시간 분석되는 영상 시스템에 어떻게 DB에 데이터를 저장해야 스토리지를 효율적으로
사용할 수 있을까요? (사람 인원, 특정 시간에 붐비는 구간, 버스 정류장에 몇 명 있는지
판정)
: 특정 구간에 사람이 등장하면 코드번호를 생성시간과 함께 부여하고, 해당 사람이 특정
구간에서 이탈하면 이탈시간을 부여합니다. (이탈의 조건은 특정 구간 테두리를 통과합니다.)
그리고 1 분단위, 또는 5 분 단위 등 정해진 시간 단위로 지나간 영상을 분석합니다.
분석 작업은 해당 시간 동안에 등장한 사람이 모두 이탈했다는 조건을 만족하면 해당
시간의 영상을 삭제하여 저장공간을 확보합니다.

22. 포인터가 필요한 이유 무엇인지 기술
: 구동된 함수를 벗어나면 스택에 저장된 변수가 사라집니다. 다른 함수에도 원하는 정보를
호출하기 위해는 동적 메모리에 정보를 저장해야 하는데, 이 정보를 호출기 위해 주소값을
저장한 포인터를 사용하여 동적 메모리에 저장된 정보를 불러옵니다.

23. 함수들이 자신만의 개별적 공간인 Stack을 사용합니다. 함수들끼리 서로 Stack을 공유하지
않는 이유에 대해 기술
: Stack을 공유하기 위해서는 함수 구동이 종료되더라도 그 정보는 남아 있어야 합니다.
메모리 공간은 한정적이기 때문에, 간단한 내용의 코드라면 가능할 수 있지만, 내용이 점점
방대해진다면 필요로 하는 메모리가 커지고 결국 부족한 상황이 발생합니다.
한정적인 메모리를 효율적으로 사용하기 위해서는, 필요한 정보만을 동적 메모리에
저장하여 활용을 하고, 나머지들은 함수가 종료되면서 메모리를 반납해야합니다.

33. 처음에는 Domain 분리가 필요 없다 생각하였던 특정 Entity 내부 어떤 멤버 변수가 시간이

지남에 따라 내부 변수 수행하는 작업의 복잡도가 높아진다면 어떻게 처리하겠습니까?

: backlog 관점에서 각 기능이 5 개 이내로 관리한다는 기준을 세웁니다. 작업의 복잡도가

높아지면서 기능이 많아진다면, 이를 분리하여 관리합니다.