<프로젝트 회고>

~일기쓰듯이, 블로그쓰듯이, 누가볼거라고 생각 ㄴㄴ ~

[자바 프로젝트]

프로젝트 첫날 (03/07 화)

대망의 첫 프로젝트. 두달간 책과 예제만 따라 치며 공부하던 나에게 처음으로 무언가를 만들어야하는 프로젝트를 시작했다. 개발자가 되기로 결심한 후 처음으로 진행하는 프로젝트이다. 사실 자바공부가 잘 맞기도 하고 정말 열심히 공부한만큼 실력적으로도 자신 있었고 그동안 꽤 많은 대외활동을 해오며 그룹 활동에 익숙해져있고, 또 잘해왔다고 생각한 만큼 협업 활동도 부담되지 않았다.

첫 프로젝트 주제는 자바의 객체지향개념을 활용한 테니스 계수기 구현, 구체적으론 ‘인터페이스’ 개념을 활용한 테니스 구현이다. 사실 객체지향쪽은 실제 코딩구현에 활발하게 사용했다기보단 이론수업, 개념정리 위주로 수업 및 공부를 진행해왔어서 막상 이 객체지향이라는 개념을 코드로 구현하자니 막막하기만 했다. 강사님이 보여주신 전기수 완성품들을 보면 짧은 기간에도 너무 잘해놔 막막했다. 우선 요구분석, 즉 테니스 계수기의 동작 프로세스를 파악하는게 우선이라 수업이 끝난 후 팀원들 각자 규칙 분석을 한 후 저녁에 디스코드로 회의를 했다.

이런 팀 활동은 서로가 편할수록 더욱 완성물 자체에 열정을 가질 수 있고 의견도 편한 분위기인걸 알기에 처음 팀이 짜인 그날부터 팀 분위기를 풀려고 많이 노력했다. 대부분의 수업이 온라인으로 이루어져 사실상 얼굴을 많이 못볼 수 있었는데 내가 주도해 자진해서 오프라인으로 수업을 나오게 하고 계속 밥도 같이 먹자고하고 다같이 모여 스터디도 매일해 정말 많이 친해졌다. 수업이 끝나고도 새벽까지 다같이 공부하며 함께 의욕을 얻으며 공부할 수 있었다. 이렇게 친해진 사람들은 학원수료를 앞둔 지금까지 가장 친한 사람들 중 하나라 사실 초반에 분위기 풀려고 애쓴건 조금 창피하긴했지만 잘했다고 생각이 들었다.

팀원들끼리 많이 친했기 때문에 디스코드 같이 음성만으로 회의를 하는게 전혀 어색하지도, 불편하지도 않았다. 그 이전까지 매일까지 이걸로 스터디를 해 편한것도 있었을듯. 다른팀 얘기를 나중에 들은건데 실제로 다른팀들은 줌이나 디코에서 한마디도 안해 결국 오프라인으로 모일 수 밖에 없다고 들었다. 이걸로 팀원들간 좋은 관계를 유지하는게 일적으로도 큰 도움이 된다는 걸 알 수 있었다 너무 노는 분위기만 아니면 ! 왜냐하면 결국 팀 프로젝트는 한사람이 다하는게 아닌 소통을 통해 할 수 있는거고 소통은 상대방이 나의 말에 집중해줄 거라는 전제에서 나오는거니까.

첫 회의가 시작되고 서로 파악한 테니스 규칙을 점검한 후 다음 단계를 진행하고자 했지만 막혔다. 단순히 자바 기본문법으로 테니스 계수기를 구현하는게 아닌 객체지향개념을 활용해야해 어려움을 느끼는 것 같았다.

하지만 다른 사람들이 지루하고 어렵게만 느껴졌던 개념수업을 오히려 나 같은 경우 문과식 공부가 익숙했던 탓인지 굉장히 재밌게 들었었고 내가 아는 개념을 직접 손으로 정리하는걸 굉장히 좋아하는 나는 이부분도 굉장히 꼼꼼하게 복습을 해두었다.

다들 감을 잡고있지 못할 때 나는 우선 내가 알고있는 객체지향의, 특히 자바 인터페이스에 대한 개념과 그 필요성, 활용방법을 한번 더 점검하고 화면공유를 통해 두가지 설명?을 하며 회의를 진행했다.

첫번째로 진행한 건 우선 인터페이스의 개념 및 그 필요성이다. 내가 수업내용과 교재 ‘자바의 정석’을 보며 공부했던 내용을 손필기로 정리해둔 자료를 공유하며 차근히 설명했다.

텍스트, 폰트, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

우선 인터페이스는 ~~ 이고 그렇기에 이 인터페이스에는 메서드의 선언부만 올 수 있다. 이러한 인터페이스가 필요한 이유는 여러가지가 있지만 구현부와 실행부의 역할분담에 큰 목적이 있다. 만약 단순히 클래스에 메서드를 정의하고 이를 실행부에서 사용하는 구조라면 실행부를 개발하는 쪽에선 해당 클래스의 메서드가 완성되기 전까진 오류가 나 개발할 수 없다. 이러한 상황을 방지하기 위해 선언부와 구현부를 나누어 해당 메서드를 사용하는 클래스에선 해당 메서드의 선언부만으로도 메서드 실행이 가능해 프로그램 테스트가 가능하다. 이렇게 이런 메서드를 사용해 만들어 만든 클래스 A를 실제 main함수에선 객체로 생성하고, 인터페이스 참조변수로 구현부 클래스인 클래스 B를 객체로 생성해 실행시 동적으로 제공한다. 클래스 B는 인터페이스 I가 구현된 것이므로 I 타입 참조변수에 대입 가능하다.

이렇게 대략적인 객체지향 개념과 인터페이스의 사용법을 설명한 후, 가장 중요한 이를 어떻게 우리 프로젝트에서 활용할지에 대해 의견을 냈다. 이를 설명하기 위해 학원에 오기전 다양한 대외활동에서 아주 잘 활용한 여러명이 함께 편집할 수 있는 ‘구글 독스’라는 공유문서를 사용했고 이는 자연히 우리 프로젝트 전반적인 공통문서가 되었다. 물론 실제 개발자들은 깃허브나 공유 노션 같은 조금 더 고기능의 툴을 사용하겠지만 당장 개발을 진행해야하는 우리에게 누구나, 바로 쉽게 사용가능한 도구를 적합하게 골랐던 것 같다. 이를 통해 물론 고기능의, 유명한 멋진 툴을 이용하는 것도 중요하지만 일에서 불편하지 않더라면 멋지진 않더라도 가능하면 최대한 쉽고, 간단한 도구를 사용해 일을 진행하는 것이 중요하다고 생각이 들었다. 방법은 쉬울수록, 누구나 알아보기 쉬울수록 좋다. 이후 다른 팀이나 이후의 프로젝트에서도 이를 사용해 개발을 진행했다는 얘기를 듣고 뿌듯했다. 이를 통해 개발자는 항상 새로운 앱이나 기술, 기능을 배우고 활용하려는 자세가 업무 능률을 훨씬 올려줄 수 있다는걸 알았고 이렇기에 개발자는 평생 공부를 해야하는 직업이겠구나~ 하는 감도 왔다.

나는 다음과 같이 프로그램, 즉 클래스 구조를 짰다.

**A class 메서드 기능 구현( interface 구현)  {**

 pointWinner(int p)  {   /\*~ 코드 구현 ~ \*/ }

// 이 메서드는 정말 어떤 선수의 포인트를 증가시키는 역할만 !

//게임 스코어 및 세트 스코어 계산=> 요건 이 메서드가 아니라 계수기 클래스에서

// \*로직

1. main에서 1 또는 2를 매개변수를 받음 (입력)
2. 1- > player1이 이긴거임 / 2-> player2가 이긴거임

dispScoreBoard() : 현재의 스코어보드를 화면에 출력한다.

{   /\*~ 여기를 이 인터페이스 구현부 클래스에서 자세히 구현하면 좋을듯? ~\*/    }

새로운 메소드 : 전체 결과 출력 => 파일에

**}**

**I  interface 테니스 규칙 {**

// 추상메서드

① pointWinner(int p) : p의 값이 1인가 2인가에 따라 해당 선수의 포인트를 증가시킨다. 포인트 득점에 따른 게임 스코어 및 세트 스코어 계산은 아래의 규칙에 따른다.

* 매개변수 : int p > 랜덤으로 1,2 고르기 (random -> main에서 입력 )
* 기능 :
* 리턴값 :

② dispScoreBoard() : 현재의 스코어보드를 화면에 출력한다.

* 매개변수 : X
* 기능 : 선수 이름, 포인트 현황(+ 듀스여부) , 게임 현황(+듀스여부), 세트 현황을 출력한다
* 리턴값 : void 출력)

/ 요런 기능 => 구현은 클래스 A에서 ? {

* 포인트 : 몇 대 몇
* 게임 : 몇 대 몇
* 세트 : 몇 대 몇

1. 새로운 메소드 : 전체 결과 파일에 출력

**B class Counter (interface의 메서드 사용) {**

pointWinner로 넘겨받은 승자정보 (누가 이겼는지 포인트 1 증가시켜서)

=> 이걸 기준으로 게임 현황에 반영

=> 즉 현황 클래스

// 포인트 현황에 반영

* 듀스 고려

=> 일정 조건 만족시 게임 ++

// 게임 현황에 반영

* 듀스 고려

=> 일정 조건 만족시 세트 ++

// 세트 현황에 반영

* 매치 달성?

=> 일정 조건 만족(과반수 이상 승리) 시 매치 : 승자 도출

**}**

**B-I-A**

**class 인터페이스 실행  {**

**main() {**

**// 게임 세팅**

1. 세트수 입력
2. 이름 입력

**// 객체 생성**

 두 선수의 이름을 입력하여

 세트수, 선수 이름 계수기 클래스의 객체를 생성한다.

계수기 b = new 계수기 (세트수, 선수이름1, 선수이름2 )

**// 게임 진행**

그 다음 포인터를 획득한 선수를 입력받은 후,

스코어보드를 계산하여 출력하는 과정을 경기가 끝나 최종 승리자가 결정될 때까지 반복한다.

=> 반복을 main에서 구현

while {

random으로 => 1 or 2

// 이긴 사람 뽑고 -> 점수 올리고 -> 게임, 세트에 반영 -> (매치가 나면 결과 도출)

**계수기 객체 . pointWinner(int p)** ;

// 계수기의 메서드를 호출해 매개변수(승자를) 넘겨주고

=> 그 포인트가 올라간 승자의 게임, 세트 점수 반영  : 계수기 클래스에서 처리

// ‘현재의’ 스코어보드 출력

한 포인트 획득할 때마다 계수기 클래스의 구현 결과를 출력

**계수기객체 . .dispScoreBoard();**

} // while

**// 최종 게임 결과 출력 (메인에서 만들어줄 기능? 또는 메서드? )**

   승자가 결정되면 최종 스코어보드와 함께 승자의 이름을 출력한다.

   => main에서 그냥 코드로 구현

      파일 입출력 ( 콘솔에 바로 띄우는게 아닌 파일 출력으로 결과 보여주기 )

}

이렇게 크게 점수를 올리고 게임현황을 출력하는 두 함수를 인터페이스에 선언부로 구현하고, 해당 메서드를 구현하는 class A와 해당 메서드를 사용해 실제로 경기를 진행하는 class B를 선언하고 main 함수에선 실제 선수정보를 입력받아 게임 조건을 세팅하고 해당 계수기 객체를 생성하면서 생성자로 A 클래스를 넣어주어 게임을 반복문으로 진행해 결과를 도출하고 최종결과를 출력하는 로직을 짰고, 해당 로직을 충분히 설명한 후 일단 이 방향으로 개발을 하기로 진행되었다.

역할분담은 원리대로 클래스 a (메서드 구현부)와 클래스 b(메서드 사용부)로 나누어 진행하기로 결정한 후 회의를 마쳤다.

실제로 내가 공부한 개념으로 아직 모호한 설계단계이지만 무언가 만들어지고 있다는 느낌이 들어 굉장히 벅찼던 기억이 난다. 개발자로서 첫발을 뗀 느낌. 회의를 마무리한 후 팀원들 중 한명이 자처해 우리가 회의한 내용을 바탕으로 클래스 구조 및 메서드 설명을 작성해주었다. 첫날 이렇게 대략적인 계획을 짠 후 하루를 마무리했다.

프로젝트 둘째날 (03/08 수)

다음날은 하루종일 프로젝트만 하는 날이라 전원이 학원에 등원해 회의하기로 했다. 등원한 후 나는 한번 더 어제 세운 로직을 설명한 후 역할을 나누어 각자 코딩구현을 시작했다. 이날 느꼈던 어려움은 총 두가지였다. 협업과정에서 어려움 하나와 기술적 어려움 하나였다.

우선 협업과정에서 어려움을 느꼈던 부분은 조장 오빠와의 의견차이였다. 시간이 한정된만큼 우리가 처음부터 구현하기보단 기존에 완성되어있는 다른 코드를 분석해 활용하자는 의견이었지만 해당 코딩은 객체지향개념을 활용하지 않은 코드라 오히려 이를 변형하는게 더 어려울 것 같다는 의견차이가 있어 고민을 하던 도중 우선 조장오빠는 어느정도 실력이 있는 전공자이기에 나머지 인원들이 어제 회의한 내용을 바탕으로 개발을 진행하고 그 오빠는 조사한 코딩을 분석하는 방향을 제안했다. 이렇게 의견을 냈던 이유는 첫째로 한 클래스를 구현하는데 4명씩 필요가 없었고. 만약 이 개발방향이 실패했을떄 도전할 플랜 b를 마련하기 위함이었다. 둘중 하나라도 성공하면 그걸로 하자는 식! 하지만 돌이켜보면 조장이고 전공자인만큼 기술적인 부분을 계속 봐줘야하는 역할인데 단독행동을 하면 나머지 조원들의 코딩이 진척이 잘 되지 않았을 위험이 있어 좋은 해결방안은 아니었던 것 같다. 그렇다고 시작도 하기전에 어느 한쪽 의견을 배제할 수는 없었기에 지금 생각해보면 정한대로 우선 따로 개발을 진행하되 그날 오전까지라던지 시간제한을 걸어두고 조금 더 진척이 보이는 쪽에 모두가 집중하였다면 조금 더 여유있게 개발이 진행되었을 것같다는 생각이 있다.

두번째 문제는 기술적인 부분이었다. 전날 계획했던 인터페이스, 구현 클래스, 메서드 호출클래스, 메인함수 클래스, 총 4개의 클래스를 구현하려다보니 아직 개발공부를 시작한지 얼마 안된 우리로서 객체의 주입을 실제로 구현하기가 어려웠다. 인터페이스에 선언된 메서드를 활용해 실행클래스에선 경기의 조건과 상관없이 점수만 계산하는 클래스 로직을 구현해야하는데, 이를 다시 객체로 생성해 메인 함수의 게임세팅과 연결지어주는게 당시 기술수준으론 어려웠다. 객체의 주입이라는 개념도 제대로 잡혀있지 않았고 인터페이스 참조변수도 활용하기가 여려워 실제 실행부와 해당 메서드를 사용하는 사용클래스의 구분 및 연계가 원활이 되지 않았다.

개발 시간이 촉박했던 시점이라 굉장히 아쉬웠지만 원래 설계했던 로직을 수정해 메서드 사용클래스를 따로 만들지 않고 실제 main함수에서 직접 메서드를 호출해 세팅된 게임조건에 따라 점수를 반복적으로 올리고 출력하는 방향으로 로직을 설계했다.

이렇게 하니 물론 구현자체는 메서드를 바로 호출만 하면되어 구현자체는 편해졌지만 결국 main함수에 게임세팅과 계수기 로직 구현, 출력까지 모두 구현해야해 객체 지향의 장점인 모듈화가 잘 이루어지지 않고 하나의 클래스에 여러 기능이 섞여 과하게 복잡한 코드가 남게되었다. 조금 더 시간적 여유가 있었다면 객체지향의 장점을 최대한 살려 각자의 기능에 집중하는 클래스 구분을 명확히 하고 그 주입, 연계관계를 통해 체계적인 프로그램을 만들 수 있었을 것 같아 팀원들과 나도 많이 아쉬워했다.

메서드 구현부는 작업을 마무리하고 있었고, 호출부와 메인함수를 같이 진행하기로 의견을 맞춘 후 회의를 마쳤다.

프로젝트 셋째날 (03/09 목)

개발 마지막 날이었고 메서드 구현부 작업을 마친 우리는 전원 메인함수 기능 구현에 집중했다. 마음이 급했고 각자 진행하던 코딩이 있었던 만큼 main 구현에 있어 의견차이가 있어 세팀 정도로 나누어 각자 main함수를 구현하기로 결정하였다. 아직도 이 방향이 옳은 방향인지는 모르겠다. 원칙적으론 서로 다른 파트를 맡아 역할분담을 하거나 하나의 방향을 정하고 그 방향이 되게끔하는게 맞았는지, 아님 결과적으로 되는걸 살리면 되니까 그렇게 나눠서 하는게 맞았을지는 아직도 잘 모르겠다. 오히려 한 사람이 화면공유를 통해 진행하고 그 코딩을 보며 의견을 내는 형식이었다면 각자 하는것보다 진행 속도가 나왔을 것같고 막혔던 부분의 해결책으 찾을 수 있었을거라는 생각이 들지만 동시에 해당 방향이 잘못된 거였다면 플랜 b 없이 단체로 시간을 날린셈이되어 구현을 하지 못했을거다. 나눠서하는 목적은 되는걸 살리자는 취지였지만 2명씩 나눠서 하다보니 한번 막히면 그 해결책이 단체로 하는것보다 훨씬 더디게 나왔고 서로의 진행상황을 알지 못해 각 팀마다 단독적으로 진행하게 된다는 단점이 있었다. “되는쪽을 밀자” 해결방식이지만 한정된 인원의 분담은 오히려 그 어떤방향도 안될 수 있다는 위험도 분명히 존재했다. 이에 대한 해결책으로 두가지를 제안했다. 첫번째는 한사람, 그러니까 조장이 각팀의 상황을 점검하고 기술적으로 도움을 주고, 무엇보다 각자의 코딩에서 활용할 코딩을 가져다 쓰면서 구현을 해 결과적으로 각 팀의 구현부분을 합치는 역할을 하여 각자 되는부분을 합치는 방식이 있고 , 어느정도까지 각자 개발하다가 일정 시간이 되면 가장 가능성이 있는 방향에 전원이 집중하는 방식을 제안했고 이 두 방법을 모두 사용하기로 하였다. 결과적으로 각자의 코딩으 ㅣ장점을 가져다 쓴 조장오빠가 깔끔하고 좋은 코딩은 아닐지라도 계수기 구현해 성공했고 우리는 아슬아슬하게 시간내에 코딩을 완료할 수 있었다. 아쉬웠던건 다들 욕심이 있던 만큼 어느정도 시간이 지나도 자신이 하던 코딩을 포기하지 못했고 결국 한쪽에 집중하자는 방안은 잘 실현되지 못했다. 다들 첫프로젝트인만큼 욕심도, 의욕도 많았기에 당연했지만 나를 포함해 안되는건 과감하게 끊고 다른 단계에 집중하는 자세가 필요했을 것 같다. 그렇게 코딩을 마무리한 우리는 현주언니의 주도로 ppt를 완성하고 새벽 1시쯤 개발을 마쳤다.

발표는 수월하게 마쳤고 나머지 두팀은 모두 2차원 배열을 통해 코딩을 구현했다. 나중에 알고보니 짧은 시간에 구현이 어려웠던 두팀 모두 깃허브에서 코딩을 찾아 변형했다고 들었다. 당시에는 직접 우리손으로 구현한 우리가 더 잘한 것 같고 대단한 것 같았지만, 구글링도 실력이라고 실제로 효율적이고 짧은 시간에 좋은 코딩을 사용해서 구현할 수 있다면 해당 자료를 잘 찾아내 분석하고 변형하는 방식이 실제 실무에서 효율적으로 일처리를 한다면 더 맞는 방향같다. 물론 우리는 공부중이니 이렇게 일일히 직접 구현하는게 더 바람직하고 효과가 있을지 몰라도 한정된 시간안에 최대한 효율적으로, 쉽게 할 수 있는건 최대한 쉽게 , 이용해 일을 처리하는 실무에선 이렇게 정석적으로ㅛ? 교과서적으로 직접 구현하는게 옳지만은 않은 것 같다.

---

전반적인 프로젝트 회고

---

첫 프로젝트였던만큼, 또 팀원들이 끈끈했던 만큼 기억에 정말 많이 남는 프로젝트이다. 또 동시에 정말 힘들었고 개발의 쓴맛을 보았던 프로젝트이기도 하다. 사실 팀 프로젝트를 많이 해왔지만 이렇게 짧은 시간내에, 빠르게 결과를 내야하는 팀 프로젝트는 거의 처음이었고, 미완성이라도 가치가 있던 문과식 팀프로젝트, 그러니까 해낸데까지하고 못한 부분은 그냥 아쉬운~ 느낌의 팀플과 달리 이번엔 얼마나 했던 성공했냐, 안했냐로 결과가 극명하게 갈리는 프로젝트는 처음이라 심적 부담감이 굉장히 컸다. 분명한 답이 없고 더 좋은 방법과 덜 좋은 방법만 존재했던 이전의 문과식? 팀 활동과 달리 명확한 답이 존재하고 그걸 어떻게든 찾아야하는 이과식 프로젝트는 그 성격이 굉장히 달랐고 노력한 만큼, 시간을 들인만큼 좋아지는 결과라는 공식이 당연하게 성립되지 않는 개발이라는 분야가 굉장히 낯설고 힘들었다. 실제로 금방 찾아질 줄 알았던 해결책이 찾아지지 않고, 실제로 설계한 내용과 기술적 한계를 느끼며 오답을 확인하는 기분은 굉장히 낯설고 힘들었다. 얼마나 좋은 계획이나 설계가 있었던건 중요하지 않고 그래서, 이걸 실제로 구현 가능해? 실행되? 가 생각보다 중요한 분야라는걸 깨달았고 개념공부와 설계가 곧 실제 구현으로 직접적으로 연결되지 않된다느걸 알았더. 사실 하루종일 잡고 있으면 나올줄알았던 해결책이 나오지 않자 굉장히 충격을 받았고 굉장히 오랜만에 절대 풀지 못할 수학문제를 만난 느낌이라 충격이었다. “문과식 사고방식을 버려야한다” 라고 개발자의 길로 들어선 비전공자 친구들이 한 말이 많이 실감이 났다. 사실 문과식 팀 프로젝트에선 나보다 더 나은 방법이 있었을 뿐 내가 틀렸다는 생각은 한번도 한적 없다. 하지만 개발에선, 안되면 틀린거고 구현 못하면 설계는 무용지물이다. 이를 인지하고 빠르게 인정하고 다른 해결책, 방법을 빠르게 모색하는게 중요한 자세라는걸 알았다. 문과식 프로젝트처럼 안되는걸 오래 붙잡는다고 나아지지 않는다. 안되는건 아무리 오래 붙잡아도 계속 안되고 시간낭비일 뿐이다. 이를 빠르게 인정하고 다른 사람들의 해결책, 의견에 집중하는 자세가 중요한걸 깨달았다. 즉 이나이에 깨달았다고하긴 조금 창피하지만 ‘내가 틀릴 수 있다’ 라는 당연한 명제를 비로소 실감한 느낌이었다. 프로젝트가 끝난 후 나와 나의 방법에 대한 객관화를 명확히 , 빠르게 하는 것도 실력인걸 깨달았고 고집을 버리고 다른 사람의 의견에 열린 마음을 가질 수 록 더욱 좋은 해결책이 보인다는 것을 알았다.

협업 방법에 있어서도 많은 걸 깨달을 수 있었다.

가령 안풀리는 문제를 다같이 할것인가, 각자 해볼것인가 각각의 방법에 대한 깨달음과 이 둘을 조율할 수 있는 방법들을 알았고 또 역할분담과 설계와 구현 등 개발을 진행하는데 있어 어떤 방식이 더 낫고 더 별로인지 조금은 감을 잡을 수 있던 시간이었던 것 같다. 그래서 처음엔 다른 사람의 코딩을 갖다 쓰자는 조장오빠의 의견이 이해가지 않았지만 시간이 지날수록 오히려 그게 더 좋은 방법이 될 수 있었을거라는 생각이 든다. 대신 그게 쓸만한 자료인지, 활용할수 있는 코드인지 빠르게 알아보는건 결국 충분한 공부가 전제되어야하는 일이기에 기초부터 쌓아올리는 정석적인 방법이 아예 잘못되었다고는 생각되지 않는다.

가치관도 많이 바뀌었다. 한땀한땀 내손으로 직접하는게 의미있다고 생각했고 실제로 이렇게 해야만 뭔가 진행이 되는 이전 팀활동과 달리 명확한 해결책이 있는 문제식 프로젝트의 경우 이 답을 어떤 방법을 쓰더라도 더욱 쉽게, 빠르게, 편하게 해결책을 찾는 것이, 즉 기존에 이 문제를 해결한 훌룡한 사례가 있다면 이를 적극적으로 활용해 결과적으로 우리의 결과물을 더욱 훌룡하게 그걸 빨리 해내는게 중요하다는걸 알았따. 교과서 적으로, 뭔가 논리적? 당연하게 생각했던 업무 방식에 대해 점검해보게 되는 기회가 되어 아마 이런게 “이과식 사고방식”인가 하고 감을 잡게 되는 기회가 되었따. 책으로 개념을 쌓고 쉬운 문제부터 연습하면 자연스레 만드는 실력이 생긴다는 나의 당연한 공부명제도 많이 깨지던 경험이었다. 오히려 코딩을 우선적으로 하면서 필요한 개념을 그때그때 찾는 방식으로 공부한 사람들이 해당 개념에 대한 이해도 크고 적용도 바로하면서 실제 코딩실력을 많이 높일 수 있다는 것도 깨달았다.

협업에 있어 관계면에서도 많은 생각이 들었다. 분명 서로 친했던만큼 의견을 말하거나, 상황을 공유하는건 굉장히 수월했지만 동시에 공적으로 일을 한다는건 의견이 대립하고 서로에 대한 긍정적인 비판과 피드백이 오간다는 것인데, 사적으로 너무 친하다보니 서로 공적으로 일을 할 때 괜스리 서로 더 서운해하고 또 공적으로 일을 하기 어려워했다. 그니까 안친했다면 느끼지 못할 서운함이 생기는 것. 이전까지는 무조건 친할수록 소통이 잘되니 좋은거지! 라고 생각했는데 오히려 공적인 관계가 주는 장점이 분명히 있다는걸 인지했다. 일 자체에 집중하고 서로에 대한 피드백을 주는쪽에서도, 받는쪽에서도 사적으로 친한관계가 그리 도움이 되지 않는다는걸 느꼈다. 소통이 원활하게 되는 편할정도의 관계와 일에 집중하고 서로 감정 빼고 일할 수 있는 어느정도 거리가 있는 격식있는 관계, 그 균형과 조율점을 잘 찾는게 중요할 것 같고 여기에 있어 리더가 어떤 역할을 수행해줘야할지 고민해보는 계기가 되었다.

평소엔 친하게 지내는 분위기를 형성하지만 일할땐 단호하게 보여주는 모습..? 이라고 대충 예상만 할 뿐 실제로 이렇게 진행해본적은 없어 많은 경험이 필요할 듯 하다

잘했다고 생각한거

: 내 설계의 부족한 부분과 구현 불가능한점을 빠르ㅔ 인정하고 방향을 수정한건 잘했다고 생각이 든다

못했다고 생각한거

: 의욕에 앞서 의견 빨리 포기 못한거, 어찌보면 고집부린거 ! 아직 명확히 의견과 생각이 정리되지 않은 상태에서 일단 내뱉고 보는식의 의견말하기

* + 생각정리하고 말하기 또는 내가 직접 코딩해보고 되면 말하기