

1. 코딩을 거의 노베이스에서 시작했기 때문에, 처음에 따라가는데에 조금 벅찬 기분도 들었지만, 뭔가 하나씩 만들어가고 결과물이 나올 때마다 정말 기뻐던 것 같습니다. 어떤 것을 만들겠다는 목표를 정하고 그것을 만들고 유지 보수하며 배우는게 코딩 실력이 가장 빠르게 느는 길이라고 들었는데 그것을 몸소 체감하고 있습니다. 특히 강사님께서 가르쳐주시는 방식이 단순히 주먹구구식 암기 기반이 아니라 비즈니스 로직을 구현하는 최신 기법에 맞춰져 있어 정말 좋은 것 같습니다. 아직 갈 길이 멀지만, 이제 면접도 다 끝났으니 한 눈 팔지 않고 프로젝트 마무리까지 전력 질주하겠습니다! 테슬라 가겠습니다!
2. DDD를 통해 얻을 수 있는 이점은 다음과 같습니다.
 - a. 비즈니스 중심의 모델링이 가능합니다. 개발자, 도메인 전문가, 기타 이해관계자들 사이의 의사소통이 촉진되어 작업의 속도를 향상시킬 수 있습니다.
 - b. 유연하고 확장 가능한 설계를 만들 수 있습니다. DDD는 도메인 모델을 기반으로 소프트웨어를 설계하므로, 도메인의 변화에 대응할 수 있는 유연하고 확장 가능한 설계가 가능합니다.
 - c. 비즈니스 도메인에 대한 명확한 이해와 그에 기반한 모델링은 코드의 품질을 향상시킵니다. 높은 수준의 추상화와 모델 기반의 설계는 시스템을 더 이해하기 쉽게 만들어 유지보수성을 향상시킵니다.
3. TDD를 통해 얻을 수 있는 이점은 다음과 같습니다.
 - a. 각각의 기능이나 모듈에 대한 테스트를 작성하도록 강제함으로써 코드의 신뢰성을 향상시킵니다.
 - b. 새로운 기능의 추가나 기존의 기능 변경을 하기 위해 테스트를 진행 후 **commit**함에 따라 안전한 리팩토링을 가능하게 하며, 코드의 유연성과 유지보수성 및 코드의 품질을 향상시킬 수 있습니다.
 - c. 코드 전체에 대한 분석을 하는 것이 아닌 일부분에 대해 분석함으로써 디버깅 시간이 단축됩니다.
4. 실제로 DDD를 효과적으로 수행하기 위해 TDD가 수반된다면 긍정적인 시너지 효과를 볼 수 있습니다. DDD는 비즈니스 로직을 모델링하고 명시적인 언어를 사용하여 설계해야 하므로, TDD를 사용하여 테스트 케이스를 통해 모델이 예상대로 작동하는지 확인하고, 최종적으로 DDD에서 도출된 모델을 고품질의 코드로 신뢰성 있게 구현 가능합니다.
5. 프로그램을 개발하는 과정에서 팀원들과 협업할 때 중요한 사항은 다음과 같습니다.
 - a. 모두가 프로젝트 진행 상황을 명확히 파악할 수 있어야 합니다. 이를 위해서 올바른 **backlog** 작성과 **issue checking** 이 수반되어야 합니다. 그렇지 못한 경우 불필요한 작업이 이루어지거나, 세부 사항에 함몰되거나, 프로그램이 망가지는 등의 문제가 발생할 수 있습니다.
 - b. 활발한 소통이 이루어져야 합니다. 혼자 가만히 코드를 작성하는 것이 아니라 자신의 상황을 공유하고 필요한 정보를 주고 받으면서 작업을 할 수 있어야 합니다. 특히 자기가 하고 있는 작업만 할 줄 아는 것이 아니라, 다양한 기법들을 동원하여 우선적으로 해야 할 일들을 애자일하게 처리할 수 있어야 합니다.
6. **IoC (Inversion of Control)** 은 객체 지향 설계에서 의존성 주입을 통해 객체 간 결합도를 낮추고, 테스트를 용이하게 하며 유지보수성 및 재사용성을 증가시킵니다. 함수 포인터 테이블을 이용하면 인터페이스를 통해 다형성을 구현할 수 있는데, 이는 **IoC** 컨테이너가 객체의 실제 타입을 모르더라도 인터페이스를 통해 정의된 메서드를 호출할 수 있도록 합니다. 따라서 객체를 생성하고 의존성을 주입할 때, 해당 객체의 인터페이스를 통해 메서드를 호출하는 방식으로 **IoC**를 구현할 수 있습니다.

7. **virtual** 메서드는 C++에서 다형성을 구현하고 객체 지향 설계의 핵심을 이루는 중요한 요소입니다. 실제로 여러 개체(entity)들이 동일한 인터페이스를 통해 조작될 수 있도록 하는 것은 객체 지향 프로그래밍의 중요한 특성이며 **IoC**와 연관이 깊습니다. **virtual** 메서드를 활용해 추상 클래스를 정의한다면, 새로운 파생 클래스를 추가하거나 기존 클래스를 수정하지 않고도 다형성을 활용할 수 있어 코드의 유연성과 확장성을 높여줍니다.
8. 최우선적으로 수행해야 하는 것은 자신이 대응하기 어려운 문제를 공유하기 위해 망설이지 말고 **issue** 를 발행하는 것입니다. 신속하게 함께 일하는 사람들에게 상황을 알려 낭비되는 시간을 최소화하고, 적극적으로 조언을 구하며 문제 해결 방안을 모색해야 할 것입니다. 그렇지 않는다면 결과적으로 비효율적인 작업을 할 수 밖에 없을 것입니다. 해결 방안에 대한 힌트를 얻거나, 다른 사람이 해결해 준 다음이라면 '아, 해결됐네. 오케이.'하고 끝내는 것이 아니라 본인의 것으로 만들기 위해 공부를 하여 유사한 상황이 재발할 시 능숙하게 대처할 수 있도록 발전해 나아가야 할 것입니다.
9. **Backlog**를 작성함으로써 얻을 수 있는 이점은 다음과 같습니다.
 - a. 최상위 아젠다에 집중하여 작업을 하게 되며, 작업의 우선 순위를 결정하는 데에 도움이 되고, 목표를 달성하기 위한 **Success criteria** 및 **To-do list** 작성을 통해 애자일한 협업 및 커뮤니케이션이 가능하도록 합니다.
 - b. 현재의 작업 상황, 이슈 상황, 추후 해야할 작업들에 대해 신속한 파악 및 추적이 가능하여 협업 능력을 극대화합니다.
 - c. 프로젝트에서 발생할 수 있는 다양한 리스크에 대한 대비책을 세울 수 있도록 도와줍니다.
10. 팀 프로젝트에서 분업하여 만든 결과물들이 융화가 되지 않고, 결과적으로 끔찍한 혼종이 탄생할 수 밖에 없었던 가장 근본적인 이유는 '**Backlog**의 관리가 제대로 되지 않았기 때문'입니다. 모두가 최상위 아젠다를 명확히 파악하고 이에 대한 성공 여부를 확인할 수 있는 방안을 공유하고 있었다면, 더 효과적으로 **DDD**를 구현하고 좋은 품질의 소프트웨어를 개발할 수 있었을 것입니다. 예로 **C++ Board** 프로젝트에서 **UI** 에서 입력받은 정보를 **parameter**로 전달받아서 **account** 도메인에서 로그인/회원가입 작업을 처리하는 것에서, **user story**에 기반한 흐름과 각 **domain**의 핵심 역할을 파악하지 못한 채 기능부터 구현하는 것(세부사항)에 함몰되어 처참한 결과가 비롯되었습니다. 이는 기능을 구현한 뒤 **backlog**를 작성했기 때문에 벌어진 안타까운 현상이라고 뼈저리게 느꼈으며 다음 프로젝트에서 절대 반복되어서는 안 될 것이라고 생각하였습니다.
11. 기능 단위로 **backlog** 를 작성하게 되면 세부사항에 관심이 집중되고, **user story**와 **domain**과 같은 최상위 아젠다는 후순위로 밀리게 됩니다. 그렇게 되면 기본적으로 중요한 작업이 무시되거나 중요도가 낮은 작업에 초점이 맞춰져 완성도가 낮아질 것입니다. 또한 통합 과정에서 테스트를 수행할 때까지 각각의 기능들이 목적에 맞게 동작하는지 파악하기 어렵게 되고, 기능 간의 의존성이 있는 경우 작업이 병렬적으로 이루어지지 못하여 애자일 프로세스의 올바른 이행이 되기 어렵습니다.

(12~20번까지 통합)

<https://github.com/Energy-CEO/SDC-Comprehensive-Evaluation/tree/main/1%EA%B8%B0/first/janghunpark/answer/comp12>

12. .

13.

```

+-----+
| Tables_in_test_db |
+-----+
| account            |
| board              |
| velocity            |
+-----+
3 rows in set (0.00 sec)

mysql> 
11
12 CREATE TABLE velocity (
13     id INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
14     speed DOUBLE NOT NULL,
15     direction DOUBLE NOT NULL,
16     xValue DOUBLE NOT NULL,
17     yValue DOUBLE NOT NULL,
18     reg_time datetime(6) DEFAULT CURRENT_TIMESTAMP(6)
19 );

```

14. .

15. .

16. .

17. .

18. .

19. .

20. .

21. 주어진 상황에서 효율적으로 **DB**를 관리하기 위해 고려해야 할 사항들은 다음과 같습니다.

- a. 영상 데이터를 사용한다는 점에서 압축이 필요하다고 생각됩니다. 큰 용량을 차지하는 데이터의 경우 프로그램 속도 저하를 유발할 수 있기 때문입니다. 다만 명확하게 사람 수를 인식하기 위해 영상의 품질이 너무 저하되지 않도록 압축률을 조절해야 합니다.
- b. 데이터의 수명 주기를 관리해야 합니다. 실시간 데이터를 다루기 때문에 과도하게 고프레임 영상 처리를 시도할 경우 시스템이 버티지 못할 수도 있을 것입니다. 때문에 데이터를 받아오는 주기, 이전 데이터를 삭제하는 주기를 성능에 맞게 적절하게 조절해야 합니다.

22. 포인터가 필요한 이유는 다음과 같습니다.

- a. 변수나 데이터 구조의 메모리 주소를 저장하기 때문에 직접적으로 메모리에 접근 가능하도록 합니다. 또한 프로그램 실행 중에 동적으로 메모리를 할당(**malloc, new**)/해제(**free**) 가능하게 합니다.
- b. 큰 데이터 구조를 전달함에 있어 데이터의 복사를 피하고 원본 데이터를 조작할 수 있기 때문에 효과적입니다.
- c. C언어에서 배열을 관리함에 있어 포인터를 사용하면 배열의 각 요소에 쉽게 접근할 수 있고, 문자열 역시 문자 배열로 표현되기 때문에 포인터를 사용하여 쉽게 다룰 수 있습니다.

23. **Stack** 은 함수 호출에 필요한 지역 변수, 매개 변수, 반환 주소 등을 저장하기 때문에 여러 함수가 동일한 스택을 공유하게 되면 함수들이 동시에 호출될 시 **stack** 에 대한 경쟁 조건이 발생할 수 있습니다. 이 때 예측 불가능한 동작이 이루어질 수 있으며 프로그램의 안전성이 저하될 것입니다. 때문에 함수마다 고유한 **stack** 을 보유함으로써 함수 간의 독립성을 유지할 수 있습니다.

(24~25번까지 통합)

<https://github.com/Energy-CEO/SDC-Comprehensive-Evaluation/tree/main/1%EA%B8%B0/first/janghunpark/answer/comp24>

24. .

25. (24~25번까지 통합)

(26~32번까지 통합)

<https://github.com/Energy-CEO/SDC-Comprehensive-Evaluation/tree/main/1%EA%B8%B0/first/janghunpark/answer/comp26>

26.

Agenda

함포가 쏘는 탄환의 종단 속도를 계산할 수 있어야 합니다.

Success Criteria

1. 초기 조건(문제에서 주어진 상황)들을 설정해야 합니다.
2. 등가속도 운동 공식에 의거해 종단 속도를 계산해야 합니다. (주어진 공식을 응용 가능합니다.)

To-do List

1. 초기 조건을 저장하는 변수 설정
2. 등가속도 운동 공식에 파라미터를 입력 받아 종단 속도를 계산하는 함수 작성

27. .

28.

Agenda

함포가 쏘는 탄환이 종단 속도에 도달할 때까지 걸린 시간을 계산할 수 있어야 합니다.

Success Criteria

1. 종단 속도와 처음 속도, 변위를 이용하여 적절한 함수를 통해 종단 속도 도달 시간을 계산해야 합니다.

To-do List

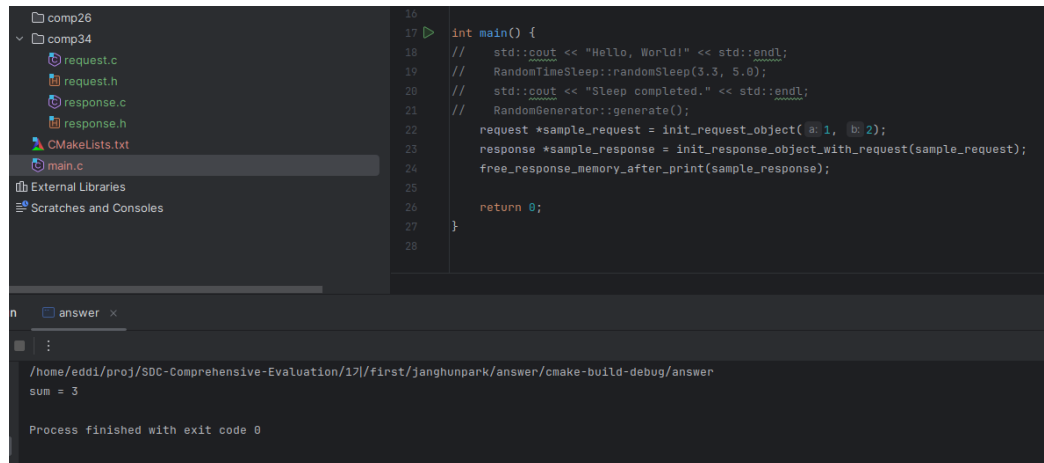
1. 종단 속도와 처음 속도, 변위에 대응하는 변수 설정
2. 위 변수들을 파라미터로 입력 받아 종단 속도 도달 시간을 계산하는 함수 작성

29. .
30. .
31. .
32. .
33. 제시된 상황에서는 **SRP** 규칙이 점차 지켜지기 힘들고 복잡성과 의존성이 늘어날 것이므로, **SRP** 규칙에 맞게 멤버 변수의 역할을 작은 단위로 분해하여 각각의 역할에 대한 클래스를 만들어야 합니다.

(34~36번까지 통합)

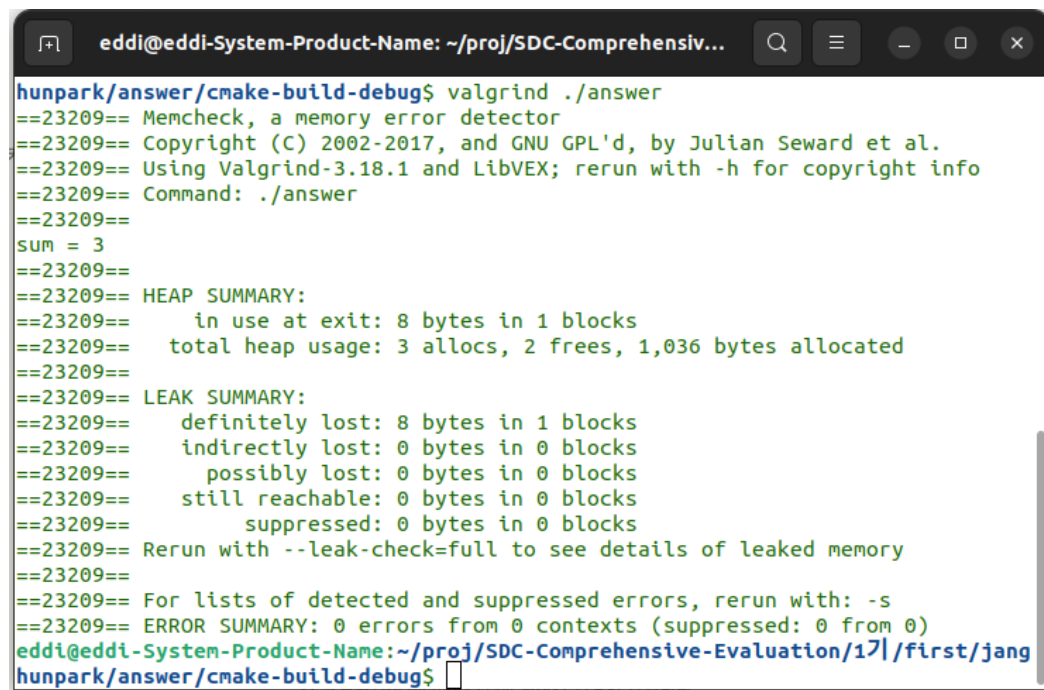
<https://github.com/Energy-CEO/SDC-Comprehensive-Evaluation/tree/main/1%EA%B8%B0/first/janghunpark/answer/comp34>

34. .
35. .
36. valgrind를 이용한 메모리 누수 체크 이미지를 첨부하였습니다.



The screenshot shows a CMake IDE interface. On the left, the project structure is visible with folders 'comp26' and 'comp34'. Under 'comp34', there are files 'request.c', 'request.h', 'response.c', 'response.h', 'CMakeLists.txt', and 'main.c'. The 'main.c' file is selected and its content is displayed in the editor. The code in 'main.c' is as follows:

```
17 int main() {  
18     // std::cout << "Hello, World!" << std::endl;  
19     // RandomTimeSleep::randomSleep(3.3, 5.0);  
20     // std::cout << "Sleep completed." << std::endl;  
21     // RandomGenerator::generate();  
22     request *sample_request = init_request_object(1, 2);  
23     response *sample_response = init_response_object_with_request(sample_request);  
24     free_response_memory_after_print(sample_response);  
25  
26     return 0;  
27 }  
28
```



The screenshot shows a terminal window with the following output:

```
hunkpark/answer/cmake-build-debug$ valgrind ./answer  
==23209== Memcheck, a memory error detector  
==23209== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.  
==23209== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info  
==23209== Command: ./answer  
==23209==  
SUM = 3  
==23209==  
==23209== HEAP SUMMARY:  
==23209==    in use at exit: 8 bytes in 1 blocks  
==23209==   total heap usage: 3 allocs, 2 frees, 1,036 bytes allocated  
==23209==  
==23209== LEAK SUMMARY:  
==23209==    definitely lost: 8 bytes in 1 blocks  
==23209==    indirectly lost: 0 bytes in 0 blocks  
==23209==    possibly lost: 0 bytes in 0 blocks  
==23209==    still reachable: 0 bytes in 0 blocks  
==23209==         suppressed: 0 bytes in 0 blocks  
==23209== Rerun with --leak-check=full to see details of leaked memory  
==23209==  
==23209== For lists of detected and suppressed errors, rerun with: -s  
==23209== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)  
eddi@eddi-System-Product-Name: ~/proj/SDC-Comprehensive-Evaluation/17/first/jang  
hunkpark/answer/cmake-build-debug$
```