Distribute Vending Machine (DVM)

OOPT 2050&60 3rd Cycle Presentation

Team 2

202111308 손승우

202111340 이수민

202111240 강찬욱

빌드 및 실행 방법 소개

- 사전준비
  - Cmake (3.15 버전 이상)
  - C++ 컴파일러 (MinGW)
  - CLion (IDE로 실행 시)

빌드 및 실행 방법 소개

- CLI로 빌드/실행 방법

```
# 1. 새 빌드 디렉터리 생성 후 이동
mkdir build
cd build

# 2. MinGW용 CMake 빌드 생성
cmake .. -G "MinGW Makefiles"

# 3. 빌드 실행
mingw32-make        # 또는 make
./main.exe          #main 실행
./MyTests.exe       #Test 실행
```

빌드 및 실행 방법 소개

- Clion (IDE) 으로 빌드/실행방법
  1. 프로젝트 열기
     File -〉 Open -〉 프로젝트 루트의 CMakeLists.txt 선택
  2. Cmake 설정 확인
     우측 하단의 Cmake 창에서 로드된 Cmake 설정 확인
  3. 빌드
     메뉴에서 Build -〉 Build Project
     빌드 로그는 하단 Build 탭에 출력
  4. 실행 설정
     상단 실행 구성 드롭다운에서 생성된 실행 타겟 선택
     Run -〉 Run ‘main’

Act. 2051. Implement Class & Method Def.

- 이전 Cycle에서 추가적으로 변경된 Class나 Method는 없었음.

Act. 2052. Implements Windows CLI

```
==============================
       2팀 DVM System
==============================
 1. 음료 구매
 2. 선결제 음료 수령
==============================
> 메뉴를 선택해주세요:
```

```
> 메뉴를 선택해주세요:1

=================== 음료 선택 메뉴 ===================
코드 | 이름           | 가격     | 수량
-----------------------------------------------------
01   | 콜라          |  1500 원 |  4 개
02   | 사이다        |  1400 원 |  5 개
03   | 녹차          |  1300 원 |  1 개
04   | 홍차          |  1300 원 |  2 개
05   | 밀크티        |  1800 원 |  6 개
06   | 탄산수        |  1200 원 |  7 개
07   | 보리차        |  1100 원 |  7 개
08   | 캔커피        |  1600 원 |  0 개
09   | 물            |  1000 원 |  0 개
10   | 에너지드링크  |  2000 원 |  0 개
11   | 유자차        |  1400 원 |  0 개
12   | 식혜          |  1500 원 |  0 개
13   | 아이스티      |  1300 원 |  0 개
14   | 딸기주스      |  1700 원 |  0 개
15   | 오렌지주스    |  1700 원 |  0 개
16   | 포도주스      |  1700 원 |  0 개
17   | 이온음료      |  1500 원 |  0 개
18   | 아메리카노    |  2000 원 |  0 개
19   | 핫초코        |  1800 원 |  0 개
20   | 카페라떼      |  2000 원 |  0 개
=====================================================

> 음료 코드와 수량을 입력하세요 (예: 01 2):
```

```
> 음료 코드와 수량을 입력하세요 (예: 01 2):01 1

[선택 완료]:
콜라 1개가 선택되었습니다.

최종 선택 목록:
- 콜라 1개

총 금액: 1500원

> 결제를 진행하시겠습니까? (예: Y or N):
```

Act. 2052. Implements Windows CLI

```
> 결제를 진행하시겠습니까? (예: Y or N):Y

결제 단계로 진행합니다...

================================
          결제 진행
================================
카드를 삽입해주세요...
11111

카드 결제 중입니다...

[결제 승인] 결제가 성공적으로 완료되었습니다.

음료를 준비 중입니다...
[음료 제공] 콜라 1개가 제공되었습니다.

================================
감사합니다. 또 이용해주세요!
================================
```

```
> 음료 코드와 수량을 입력하세요 (예: 01 2):11 1

[재고 부족] 유자차는(은) 현재 0개 남아 있습니다.

[재고 부족] '유자차'는 현재 0개 남아 있습니다.
[재고 확인 중...] 다른 DVM들에 정보 요청 중입니다...

[재고 확인 완료] '유자차 1개'는 아래 DVM에서 수령 가능합니다.
위치: x좌표 12 y좌표 12 (T2)
총 금액: 1400원

> 선결제를 진행하시겠습니까? (예: Y or N):Y
```

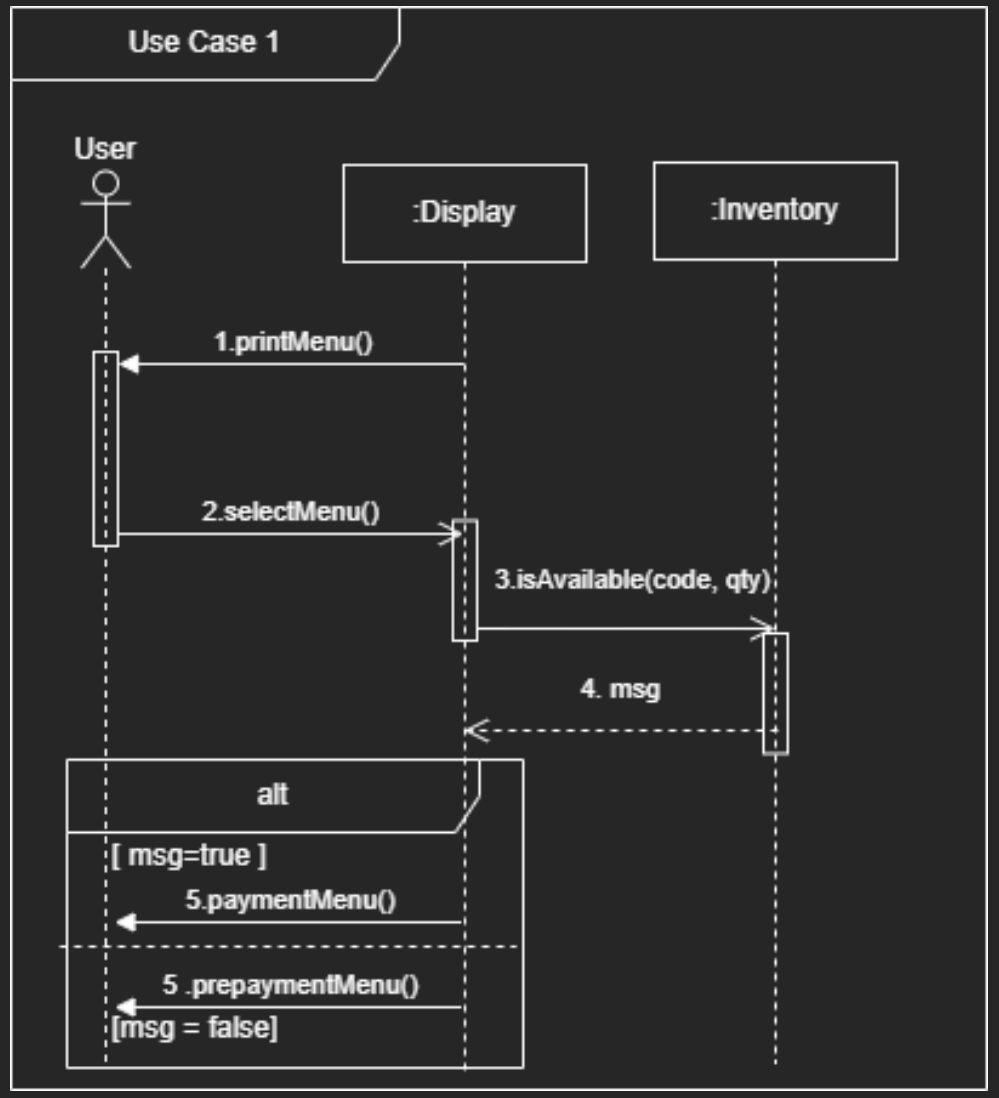
Act. 2052. Implements Windows CLI

```
==============================
          결제 진행
==============================
카드를 삽입해주세요...
11111

카드 결제 중입니다...

[결제 승인] 결제가 성공적으로 완료되었습니다.
발급된 인증코드: AnpVfiiW
수령 위치: x좌표 12 y좌표 12 (T2)

※ 해당 위치에서 인증코드를 입력하면 음료를 받을 수 있습니다.

==============================
감사합니다. 또 이용해주세요!
==============================
```

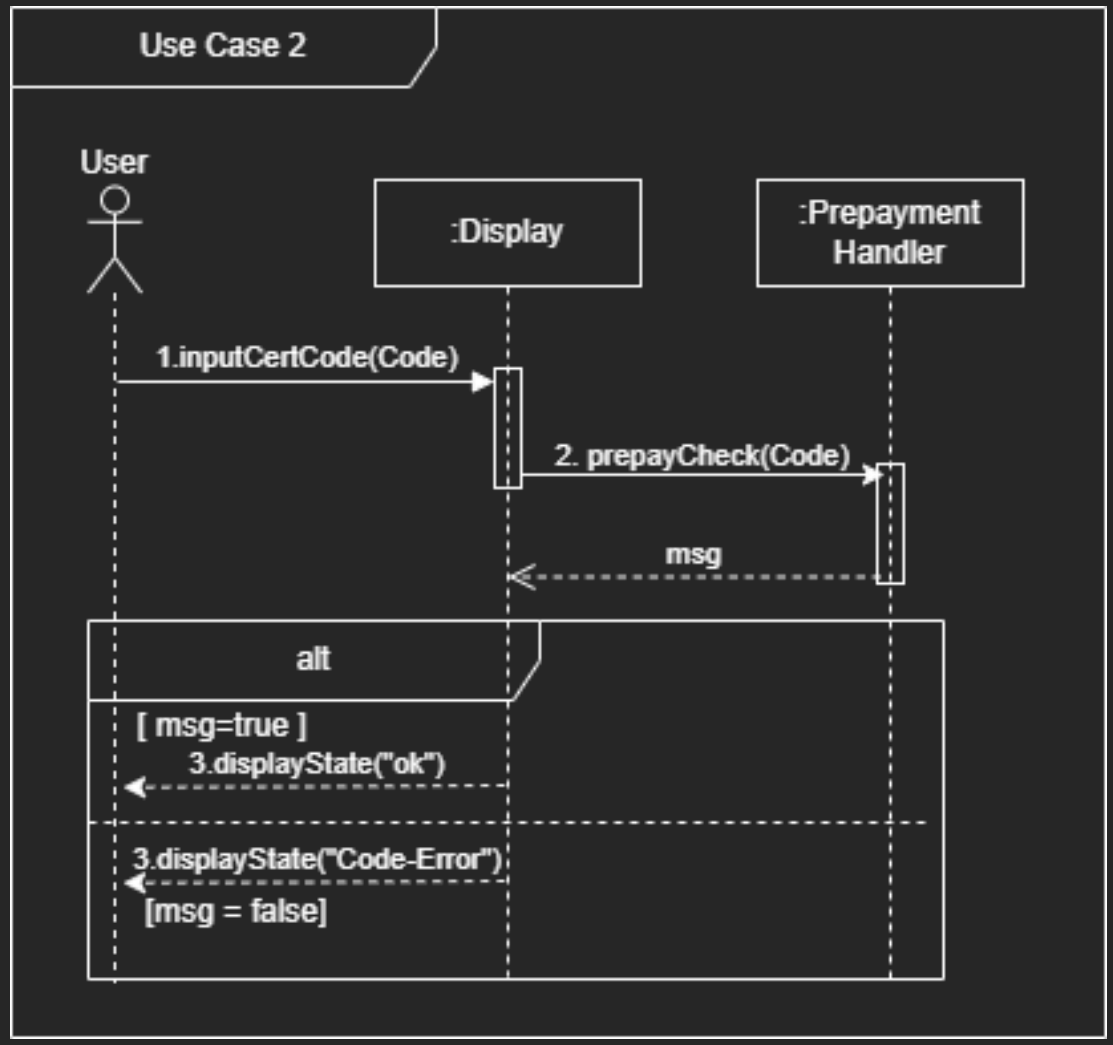
```
============================
    선결제 물품 수령
============================

인증 코드를 입력해주세요 :
AnpVfiiW

초기 화면으로 돌아갑니다.
------------------------------------------------------

음료를 준비 중입니다...
[음료 제공] 유자차 1개가 제공되었습니다.

==============================
감사합니다. 또 이용해주세요!
==============================
```

Act. 2053. Implement Reports

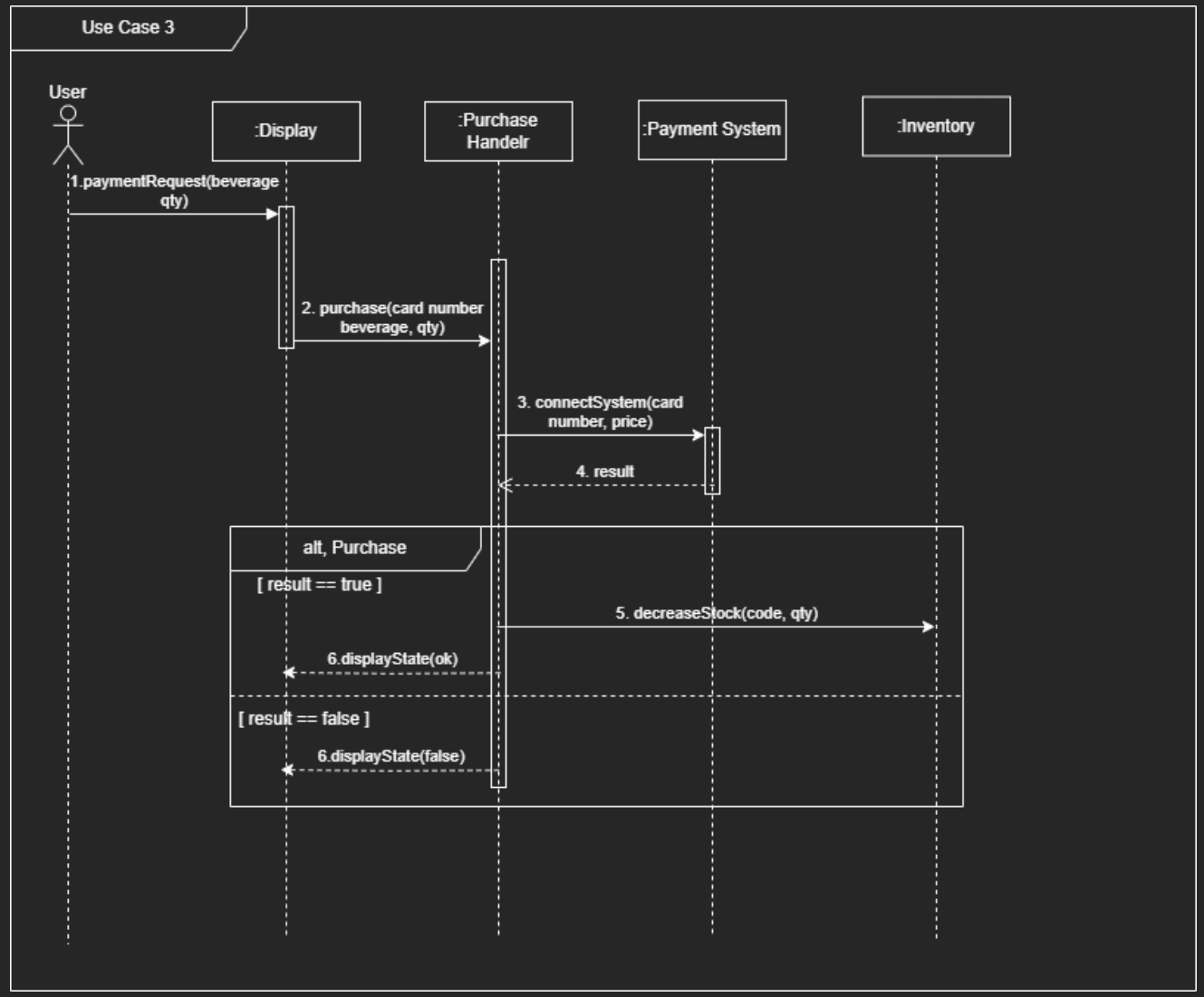
1. Sequence Diagram



Act. 2053. Implement Reports
1. Sequence Diagram

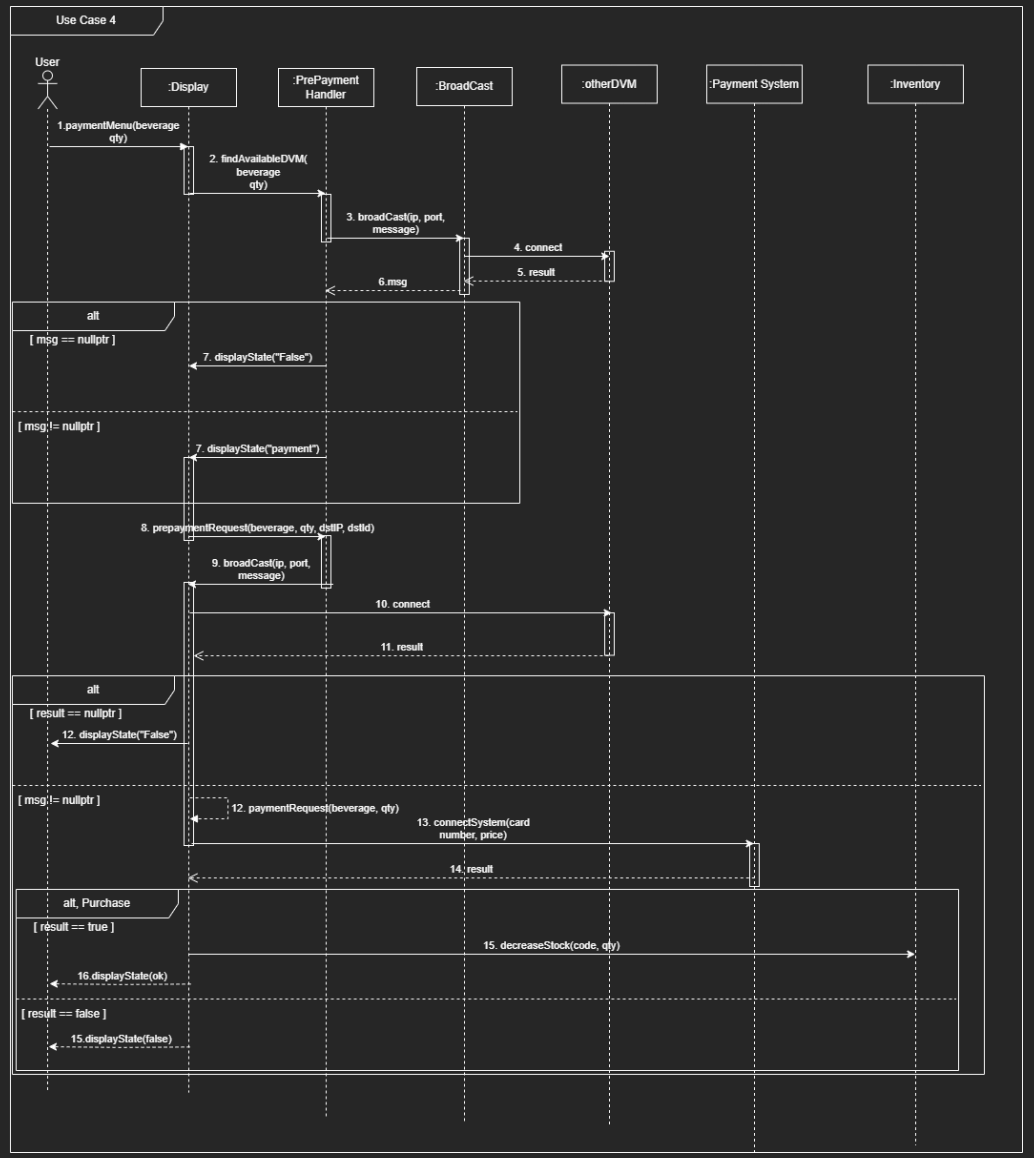


Act. 2053. Implement Reports
1. Sequence Diagram

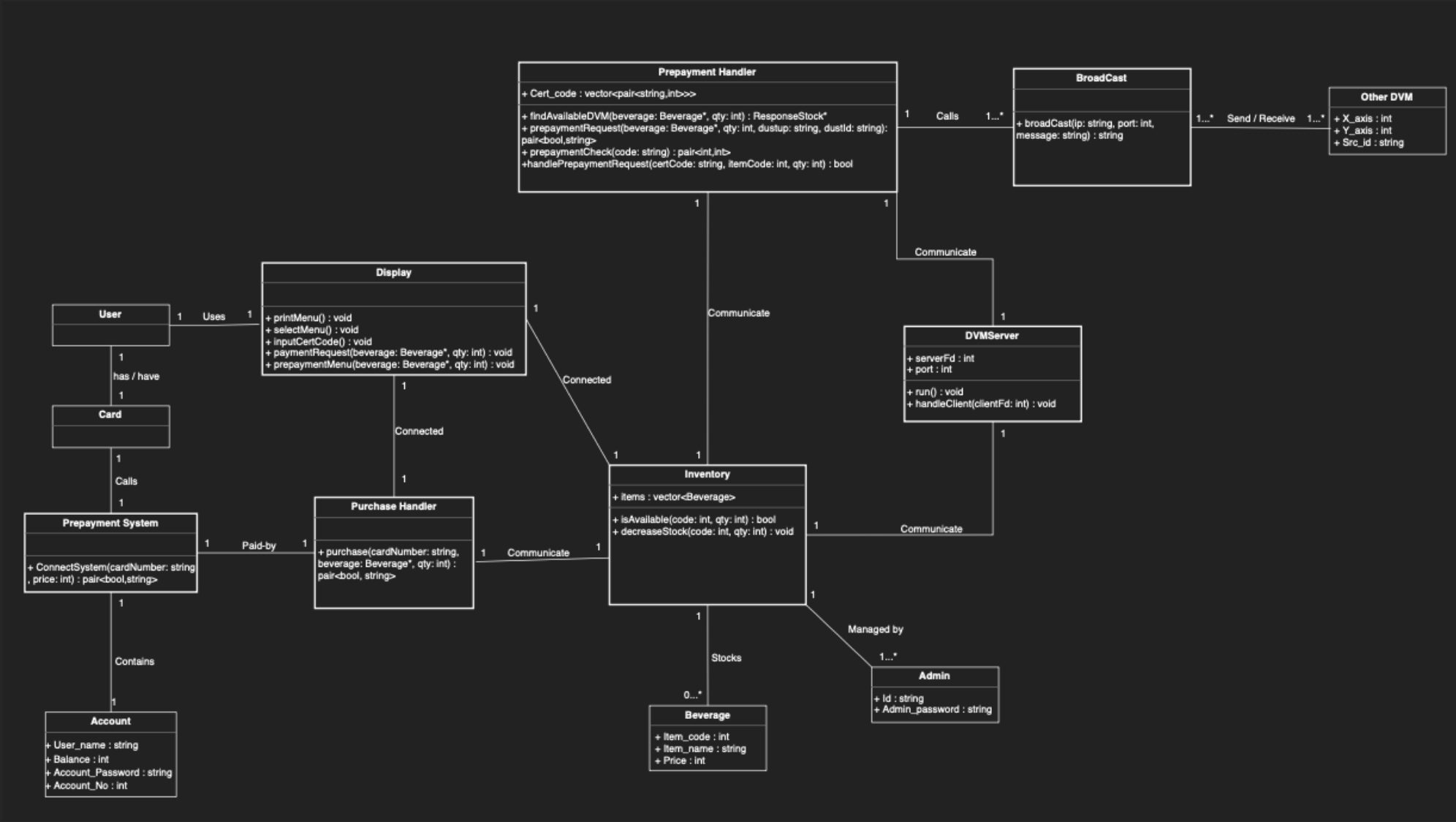


Act. 2053. Implement Reports

1. Sequence Diagram



Act. 2053. Implement Reports
2. Class Diagram



Act. 2061. Unit Testing

- Unit 테스트 대상
  - Application Layer에 존재하는 도메인 로직과 비즈니스 로직
  - 화면 출력 및 외부 시스템 연결은 제외
- Unit 테스트 방법
  - **Google Test 프레임워크**를 활용하여 주요 로직에 대한 Unit 테스트 작성
  - 실제 의존 객체 대신 **Fake 객체(FakePaymentSystem, FakeBroadCast 등)** 를 사용하여 독립성 확보
  - EXPECT_EQ, EXPECT_TRUE 등을 통해 정상 동작 및 예외 상황 검증

Act. 2061. Unit Testing(gtest)
Inventory Unit Test

Num	Test Case Name/ Related Function	Description	Condition	Expect	Pass/Fail
UT-1.1	LoadInventory_Success/ getBeverage()	BeverageRepository에서 음료 7종을 로드하여 Inventory에 정상적으로 저장되는지 확인	음료 7종 을 저장하는 FakeRepository 사용	itemCount() == 7, 각 이름 일치	Pass
UT-1.2	GetBeverage_InvalidCode_ReturnsNullptr/ getBeverage()	유효하지 않은 코드 입력 시 nullptr 반환 확인	code == 0 or 9999	nullptr 반환	Pass
UT-1.3	IsAvailable_WorksCorrectly/ isAvaliable()	충분/부족 재고 및 유효하지 않은 코드에 대해 isAvailable 동작 확인	qty == 5 (충분), qty == 15 (부족), code==99, code==0 (잘못된 코드)	true / false 정확히 반환	Pass
UT-1.4	IsValidCode_WorksCorrectly/ isValidCode()	inventory에 존재하는 음료코드인 경우만 true 반환되는지 확인	code == 100 code == -1, code == 0 code == 1, code ==2, code ==3	유효한 코드 외 false, 유효한 코드 시 true	Pass

Act. 2061. Unit Testing(gtest)
Inventory Unit Test

Num	Test Case Name/ Related Function	Description	Condition	Expect	Pass/Fail
UT-1.5	DecreaseStock_Updates_Repository_And_Inventory/ decreaseStock()	재고 차감 시 Inventory 내부 상태와 Repository에 반영되는지 확인	code == 1, qty == 3	stock 3 감소, updateCalled == true	Pass
UT-1.6	DecreaseStock_Fail_Invalid_Code/ decreaseStock()	코드가 유효하지 않을 경우 재고 차감이 발생하지 않는지 확인	code == 0	updateCalled == false	Pass
UT-1.7	DecreaseStock_Fail_Not_Enough_Quantity/ decreaseStock()	보유 수량보다 많은 수량 차감 시 실패 처리 여부 확인	code == 1, qty == 100	updateCalled == false	Pass
UT-1.8	IncreaseStock_MAX/ increaseStock()	maxStock 보다 크게 증가시켰을 시 실패 처리 여부 확인	maxStock==100, qty == maxStock + 1	메소드 호출 전 stock 과 동일	Pass
UT-1.9	IncreaseStock_Success/ increaseStock()	maxStock 보다 작게 증가시켰을 시 성공 처리 여부 확인	maxStock == 100 qty == 10	메소드 호출 전 stock 에서 10 증가	Pass

Act. 2061. Unit Testing(gtest)
Prepayment Unit Test

Num	Test Case Name/ Related Function	Description	Condition	Expect	Pass/Fail
UT-2.1	MakeRequestStockMessage_FormatsCorrectJson/ makeRequestStockMessage()	JSON 형식이 올바르게 생성되는지 검증	itemCode = 5, qty = 3	"05", "3" 포함된 JSON 반환	Pass
UT-2.2	MakeRequestStockMessage_TwoDigitItemCode/ makeRequestStockMessage()	itemCode가 두 자리일 때 padding 없이 처리되는지 확인	itemCode = 12	"12"로 그대로 출력	Pass
UT-2.3	MakeRequestPrepaymentMessage_GeneratesCorrectJson/ makeRequestPrepaymentMessage()	요청 메시지 JSON의 항목 구성 확인	certCode, itemCode=5, qty=2	msg_type == "req_prepay" 포함	Pass
UT-2.4	MakeRequestPrepaymentMessage_TwoDigitItemCode/ makeRequestPrepaymentMessage()	itemCode가 10 이상일 때 그대로 출력되는지 확인	itemCode = 15	"15" 출력	Pass

Act. 2061. Unit Testing(gtest)
Prepayment Unit Test

Num	Test Case Name/ Related Function	Description	Condition	Expect	Pass/ Fail
UT-2.5	FindDistance_ReturnsCorrectDistance/ findDistance()	거리 계산 함수의 계산 유효성 확인	DVM 위치 (3,4), 대상 (0,0)	5.0 반환	Pass
UT-2.6	GenerateCertificationCode_ReturnsCorrectLengthAndCharset/ generateCertificationCode()	생성된 인증 코드에 대한 길이와 문자 유효성 확인	length = 10	길이 = 10, 모든 문자는 유효	Pass
UT-2.7	PrepaymentRequest_Success_ReturnsTrue/ prepaymentRequest()	타 DVM에 대한 선결제 가능 여부 응답이 availability = "T“ 인 경우	availability = "T“	True	Pass
UT-2.8	PrepaymentRequest_Failure_ReturnsFalse/ prepaymentRequest()	타 DVM에 대한 선결제 가능 여부 응답이 availability = "F“ 인 경우	availability = "F“	false	Pass

Act. 2061. Unit Testing(gtest)
Prepayment Unit Test

Num	Test Case Name/ Related Function	Description	Condition	Expect	Pass/ Fail
UT-2.9	PrepaymentRequest_InvalidMsgType_ReturnsFalse/ prepaymentRequest()	타 DVM 의 msg_type이 올바르지 않을 경우의 검증	msg_type == “wrog_msg”	false 반환	Pass
UT-2.10	PrepaymentRequest_InvalidJson_ReturnsFalse/ prepaymentRequest()	타 DVM 의 요청 형식이 json 이 아닐 경우의 검증	{invalid json}	false 반환	Pass
UT-2.11	FindAvailableDVM_ReturnsClosestAvailableDVM/ findAvaliableDVM()	여러 응답 중 가장 가까운 DVM 선택하는지 검증	2개 DVM 응답, 거리 다름	가까운 DVM 반환	Pass
UT-2.12	FindAvailableDVM_AllInsufficientStock_ReturnsNullptr/ findAvaliableDVM()	모든 DVM 재고 부족 시	모든 DVM 응답에 대해 item_num 〈 request qty	nullptr 반환	Pass

Act. 2061. Unit Testing(gtest)
Prepayment Unit Test

Num	Test Case Name/ Related Function	Description	Condition	Expect	Pass/ Fail
UT-2.13	HandlePrepaymentRequest_Success/ handlePrepaymentRequest()	타 DVM의 선결제 요청 성공 시	qty = 2, 재고 충분	true 반환 인증코드 저장됨 수량 감소	Pass
UT-2.14	HandlePrepaymentRequest_StockUnavailable_Fails/ handlePrepaymentRequest()	타 DVM의 선결제 요청 시에 수량이 없을 경우	qty = 20	False 반환 인증코드 저장 X 수량 유지	Pass
UT-2.15	PrePaymentCheck_Success/ prePaymentCheck()	선결제 물품 수령 시 유효한 코드일 경우	VALID123 미리 저장.	itemCode, qty 반환 certCode삭제됨	Pass
UT-2.16	PrePaymentCheck_CodeNotFound/ prepaymentCheck()	선결제 물품 수령 시 유효하지 않은 코드일 경우	UNKNOWN 입력	-1, -1 반환	Pass

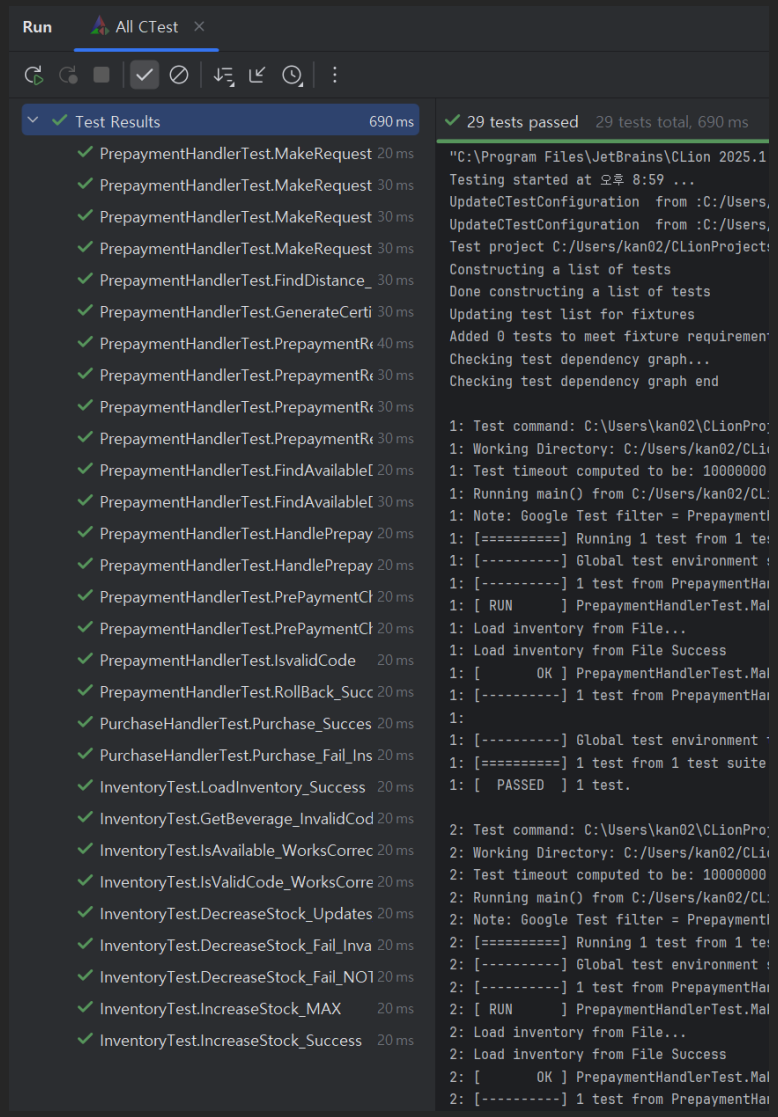
Act. 2061. Unit Testing(gtest)
Prepayment Unit Test

Num	Test Case Name/ Related Function	Description	Condition	Expect	Pass/ Fail
UT-2.17	IsvalidCode/ IsvalidCode()	인증 코드에 대한 유효성 확인	Code = aa1bB2cC Code = short Code = toolongcode123 Code = invalid!@# Code =	aa1bB2cC 일 경우 true 이 외 false	Pass
UT-2.18	RollBack_Success/ rollBackPrepaymentRequest	롤백 처리의 유효성 확인	Qty = 2	메소드에서 true 반환 Qty 그대로 유지 저장되었던 코드 삭제	Pass

Act. 2061. Unit Testing(gtest)
Purchase Unit Test

Num	Test Case Name/ Related Function	Description	Condition	Expect	Pass/ Fail
UT-3.1	Purchase_Success/ purchase()	결제 시스템이 성공 응답을 주었을 때 재고 차감 및 결제 성공 메시지 확인	카드번호: "1234-5678", 수량: 2, 결제 성공	result.first == true, result.second == "결제 성공", 수량 2만큼 감소	Pass
UT-3.2	Purchase_Fail_Insufficient Balance/ purchase()	결제 시스템에서 잔액 부족 응답을 줄 경우 재고 차감 없이 실패 처리	카드번호: "1234-5678", 수량: 2, 결제 실패	result.first == false, result.second == "잔액 부족", 수량 유지	Pass

Act. 2061. Unit Testing(gtest)
Test Result



Act. 2061. Unit Testing(gtest)
Result of Test - TC 14

Test Case 14 – Card Format Error
목적: 카드 입력값의 형식이 하이픈 누락 등으로 유효하지 않은 경우, 시스템이 적절히 처리하는지 확인
입력: 잘못된 카드 번호 (예: 하이픈 없이 1234567812345678)
흐름:
1. 사용자가 카드 번호 입력
2. 카드 포맷 검증 (regex 또는 split 기준)
3. 하이픈이 없거나 잘못된 형식 → 에러 메시지 출력

- 검증팀의 의견과 달리, 카드 입력 값이 유효하지 않을 시, 적절한 처리를 하고 있음을 확인.

Act. 2061. Unit Testing(gtest)
Result of Test - TC 14

```
==============================
        결제 진행
==============================
카드를 삽입해주세요...
11111112222222

카드 결제 중입니다...

카드 형식이 잘못되었습니다.

초기 화면으로 돌아갑니다.
--------------------------------------


==============================
       2팀 DVM System
==============================
```

Act. 2061. Unit Testing(gtest)
Result of Test – TC 15

Test Case 15 – Invalid Item Code
목적: 존재하지 않는 itemCode를 입력했을 때 시스템이 적절히 무효 처리하는지 확인
입력: itemCode = 0 또는 존재하지 않는 코드
흐름:
1. 사용자 또는 요청 메시지에서 잘못된 itemCode 전달
2. 시스템에서 getItemInfo(itemCode) 호출
3. 해당 코드가 존재하지 않으면 null 반환
이후 요청 중단 또는 에러 메시지 처리 수행 (예: 'Invalid item' 메시지 반환)

- 검증팀의 의견과 달리, item Code 범위가 다를 경우에도 적절한 처리를 하고 있음을 확인.

Act. 2061. Unit Testing(gtest)
Result of Test - TC 15

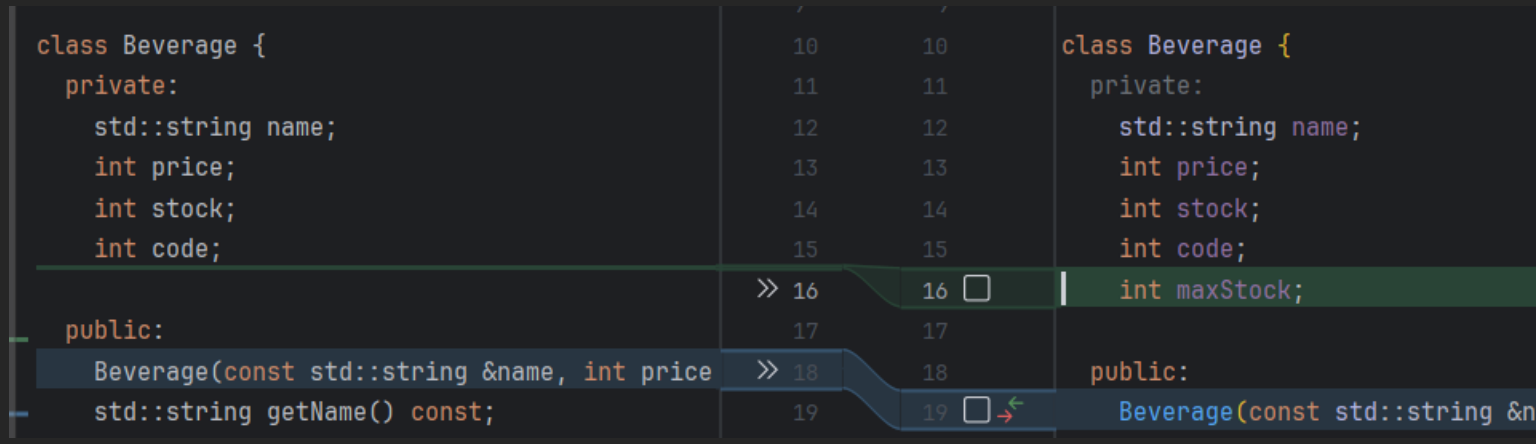
```
> 음료 코드와 수량을 입력하세요 (예: 01 2): 00 1

[입력 오류] 해당 음료 코드는 존재하지 않습니다.
다시 입력해주세요.
-----------------------------------------

> 음료 코드와 수량을 입력하세요 (예: 01 2): 0 1

[입력 오류] 해당 음료 코드는 존재하지 않습니다.
다시 입력해주세요.
-----------------------------------------

> 음료 코드와 수량을 입력하세요 (예: 01 2): 22 1

[입력 오류] 해당 음료 코드는 존재하지 않습니다.
다시 입력해주세요.
-----------------------------------------

> 음료 코드와 수량을 입력하세요 (예: 01 2):
```

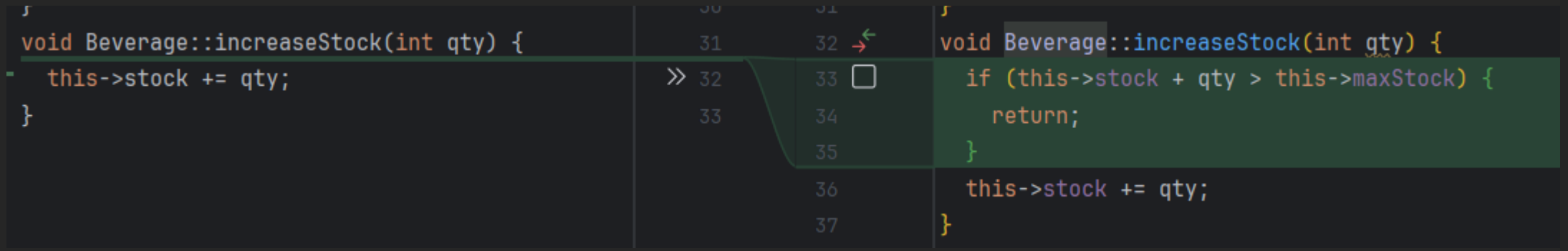
Act. 2061. Unit Testing(gtest)
Result of Test – TC 16

Test Case 16 – Inventory Overflow
목적: 인벤토리에 허용된 범위 이상으로 아이템이 추가될 때 예외 발생 여부 확인
입력: 테스트를 위한 대량 아이템 등록 (예: 1000개 이상)
흐름:
1. 반복적으로 Inventory::addItem() 호출
2. 내부적으로 배열 또는 리스트 용량 초과
3. 예외 또는 삽입 실패 반환

- Beverage 클래스에 MAX_STOCK 을 정의하여 increaseStock(int qty) 호출 시 MAX_STOCK 보다 커질 경우 반영하지 않는 로직을 추가함.

Act. 2061. Unit Testing(gtest)
Result of Test - TC 16





- Beverage 클래스에 maxStock 을 정의하여 increaseStock(int qty) 호출 시 MAX_STOCK 보다 커질 경우 반영하지 않는 로직을 추가함.

Act. 2061. Unit Testing(gtest)
Result of Test - TC 16

```
// MAX STOCK 보다 크게 증가시켰을 시
TEST(InventoryTest, IncreaseStock_MAX) {
  FakeBeverageRepository fakeRepo;
  Inventory inventory(&fakeRepo);
  Beverage* testBeverage = inventory.getBeverage( code: 1);
  int testBeverageQty = testBeverage->getStock();
  inventory.increaseStock( code: 1,  qty: testBeverage->getMaxStock() + 1);

  EXPECT_EQ(testBeverage->getStock(), testBeverageQty);
}

// MAX STOCK 보다 작게 증가시켰을 때
TEST(InventoryTest, IncreaseStock_Success) {
  FakeBeverageRepository fakeRepo;
  Inventory inventory(&fakeRepo);
  Beverage* testBeverage = inventory.getBeverage( code: 1);
  int testBeverageQty = testBeverage->getStock();
  inventory.increaseStock( code: 1,  qty: 10);

  EXPECT_EQ(testBeverage->getStock(), testBeverageQty + 10);
}
```

- 변경한 increaseStock() 가 maxStock 이상으로 저장이 되지 않는 것에 대한 테스트 코드 추가

Act. 2061. Unit Testing(gtest)
Result of Test – TC 17

Test Case 17 – Valid Code Registration
목적: 유효한 코드가 정상적으로 등록되고 사용 가능한 상태로 전환되는지 확인
입력: 형식이 올바르고 등록되지 않은 인증 코드
흐름:
1. 사용자가 선결제 인증 코드 입력
2. 코드 유효성 확인 (형식, 중복 여부)
3. 인증 코드 저장 및 사용 가능 상태 전환

- 인증코드 입력 시 저장되는 로직은 요구사항 및 Usecase 에 없으므로, 구현 대상이 아님

  (입력 시가 아닌 선결제 시 인증코드 저장)

- 이외의 코드 유효성 확인 기능은 정상 작동

Act. 2061. Unit Testing(gtest)
Result of Test - TC 17

```
==========================
    선결제 물품 수령
==========================

인증 코드를 입력해주세요 :
11111

[인증 실패] 인증 코드의 형식이 잘못되었습니다.
--------------------------------------------------

초기 화면으로 돌아갑니다.
--------------------------------------------------
```

Act. 2061. Unit Testing(gtest)
Result of Test – TC 18

Test Case 18 – Missing Menu Token
목적: JSON 요청 메시지에 필수 키(menu)가 누락되었을 때 적절한 오류 처리를 수행하는지 확인
입력: menu 키가 없는 JSON 메시지
흐름:
1. JSON 메시지 수신
2. hasKey("menu") 또는 get("menu") 시 null
3. 필수 필드 누락 → 오류 메시지 반환 또는 예외 처리

- Json 요청 메시지 형식에는 menu가 필수 키가 아니고, 요구사항에도 없으므로 구현 대상이 아님.

Act. 2061. Unit Testing(gtest)
Result of Test – TC 19

Test Case 19 – Message Overload
목적: 짧은 시간 내에 너무 많은 메시지가 도착했을 때 시스템이 과부하를 감지하고 처리하는지 확인
입력: 동시 접속 또는 메시지 폭주 (e.g., 100개 이상)
흐름:
1. 동시에 여러 메시지 수신
2. 처리 큐 또는 스레드풀 초과
3. 일부 요청 무시 또는 에러 반환 ('System overload')

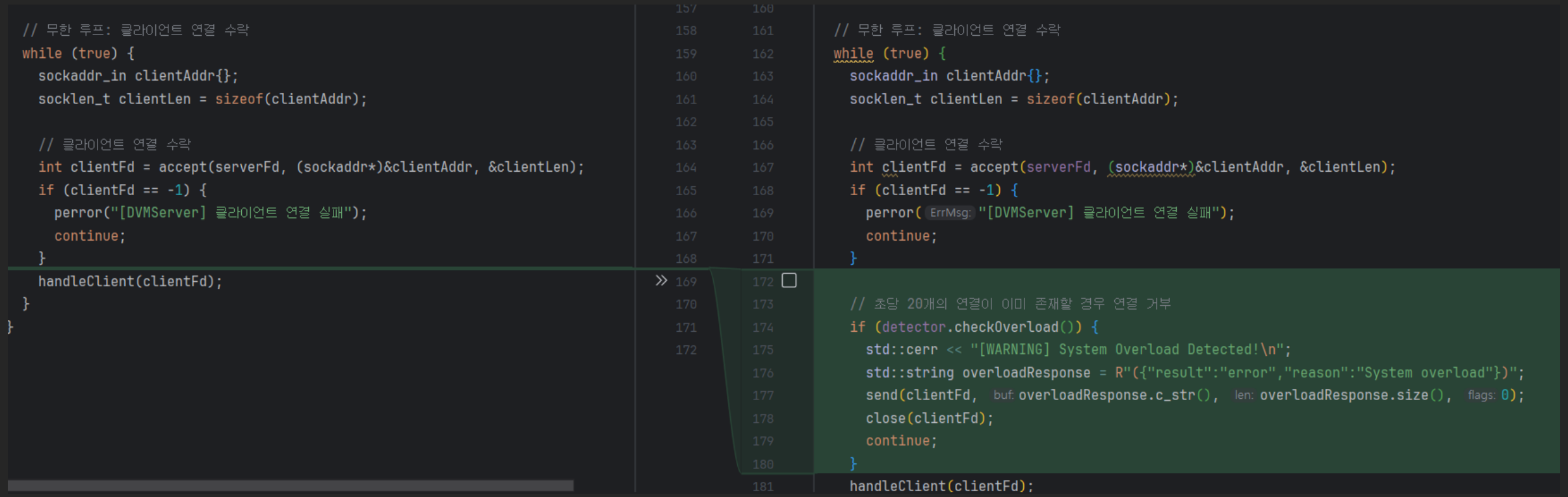
- OverloadDetector 클래스를 구현하여 DVMServer 에 1초당 20개 이상의 요청이 진입하지 못하도록 변경

Act. 2061. Unit Testing(gtest)
Result of Test – TC 19

```
class OverloadDetector {
  private:
    queue<steady_clock::time_point> timestamps;
    int thresholdPerSecond;

  public:
    OverloadDetector(int threshold) : thresholdPerSecond(threshold) {}

    bool checkOverload() {
      auto now :time_point<steady_clock, steady_clock::duration> = steady_clock::now();
      timestamps.push(now);

      // 1초 이전 타임스탬프 제거
      while (!timestamps.empty() &&
             duration_cast<milliseconds>( d: now - timestamps.front()).count() > 1000) {
        timestamps.pop();
      }

      return timestamps.size() >= thresholdPerSecond;
    }
};
```

- checkOverload() 는 thresholdPerSecond(초당 수용 가능한 쓰레드 수) 보다 현재 진입한 쓰레드가 많으면 false 를 반환

Act. 2061. Unit Testing(gtest)
Result of Test - TC 19



```
// 무한 루프: 클라이언트 연결 수락
while (true) {
  sockaddr_in clientAddr{};
  socklen_t clientLen = sizeof(clientAddr);

  // 클라이언트 연결 수락
  int clientFd = accept(serverFd, (sockaddr*)&clientAddr, &clientLen);
  if (clientFd == -1) {
    perror("[DVMServer] 클라이언트 연결 실패");
    continue;
  }
  handleClient(clientFd);
 }
}
```

```
// 무한 루프: 클라이언트 연결 수락
while (true) {
  sockaddr_in clientAddr{};
  socklen_t clientLen = sizeof(clientAddr);

  // 클라이언트 연결 수락
  int clientFd = accept(serverFd, (sockaddr*)&clientAddr, &clientLen);
  if (clientFd == -1) {
    perror( ErrMsg: "[DVMServer] 클라이언트 연결 실패");
    continue;
  }

  // 초당 20개의 연결이 이미 존재할 경우 연결 거부
  if (detector.checkOverload()) {
    std::cerr << "[WARNING] System Overload Detected!\n";
    std::string overloadResponse = R"({"result":"error","reason":"System overload"})";
    send(clientFd,  buf: overloadResponse.c_str(),  len: overloadResponse.size(),  flags: 0);
    close(clientFd);
    continue;
  }
  handleClient(clientFd);
```

- DVMServer에서 이를 이용하여 과부화를 탐지하여 과부화 시 연결 거부.

Act. 2061. Unit Testing(gtest)
Result of Test - TC 20

Test Case 20 – Concurrent Access Conflict
목적: 여러 스레드가 동시에 동일한 파일 또는 데이터에 접근할 때 충돌을 감지하고 처리하는지 확인
입력: 동일한 인증코드 저장 요청이 동시에 여러 스레드에서 발생
흐름:
1. 동시에 동일한 코드로 파일 쓰기 요청 발생
2. 락(lock) 미사용 시 race condition 발생
3. 충돌 발생 → 하나의 요청만 성공, 나머지는 에러

- DataBase 로 사용하고 있는 txt 파일에 여러 쓰레드가 동시적으로 접근할 수 없도록 Mutex Lock 을 통해 동시성 제어를 구현함.

Act. 2061. Unit Testing(gtest)
Result of Test – TC 20

```
}
std::vector<Beverage> FileBeverageRepository::lo
  std::ifstream file(filePath);
  if (!file.is_open()) {
    std::cerr << "[Error] Failed to open invento
    exit(1);
  }

  std::vector<Beverage> items;
```

```
std::vector<Beverage> FileBeverageRepository::loadBeveragesFromFile() {
  std::lock_guard<std::mutex> lock( [&] fileMutex);
  std::ifstream file(filePath);
  if (!file.is_open()) {
    std::cerr << "[Error] Failed to open inventory file: " << filePath << s
    exit( Code: 1);
  }

  std::vector<Beverage> items;
```

```
void FileCertificationCodeRepository::saveToFile() {
  std::ofstream file(filePath, std::ios::trunc);
  for (const auto &[certCode, info]: codeMap) {
    file << certCode << " " << info.itemCode << " " << ir
  }
}
```

```
}

void FileCertificationCodeRepository::saveToFile() {
  std::lock_guard<std::mutex> lock( [&] mtx);
  std::ofstream file(filePath,  mode: std::ios::trunc);
  for (const auto &[certCode :const string , info]: codeMap) {
    file << certCode << " " << info.itemCode << " " << info.itemNu
  }
}
```

- DB 파일을 읽는 FileBeverageRepository 와 FileCertificationCodeRepository 클래스의 메소드에 mutex lock 을 적용

Act. 2061. Unit Testing (gtest)
Result of Test – Brute Force Test

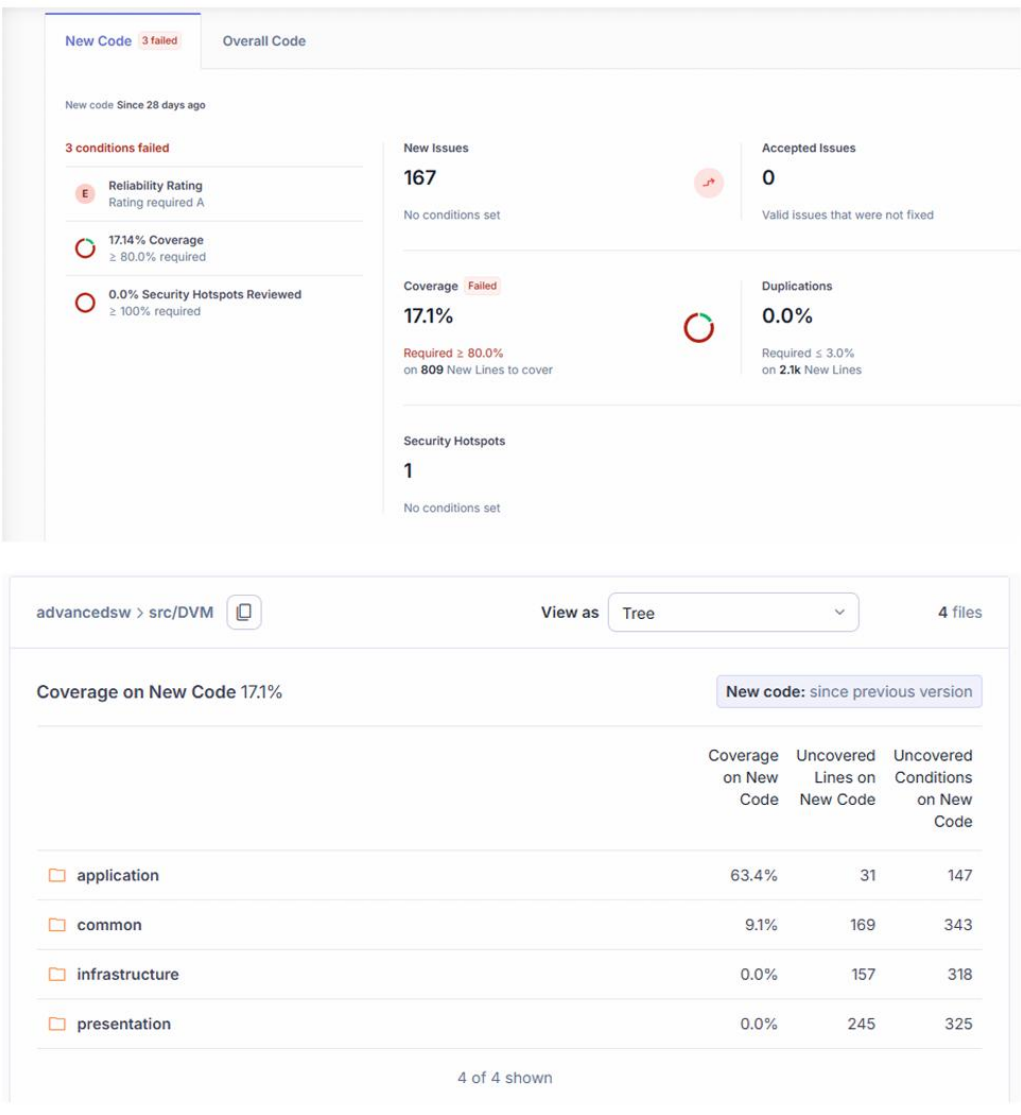
TestName	TestDescription	Expected Output
RUC 01 초기메뉴 선택	콘솔에 선택지를 보여주고, 사용자가 유효한 숫자(1 또는 2)를 입력할 때까지 반복 입력을 받는다.	1이나 2가 아니면 잘못 입력했다는 메세지와 함께 다시 입력하도록 해야함
RUC 01 초기메뉴 선택	각 번호에 맞게 입력했을 때 올바르게 출력값이 나오는지 확인	1번 입력했을 경우 음료 구매 화면으로, 2번 입력 했을 경우 선결제 화면으로 이동
RUC 02 음료 종류 및 수량 선택	유효하지 않은 번호 입력 시 결과 확인	음료 이외 번호 입력 시 오류처리 및 재입력 요청
RUC 02 음료 종류 및 수량 선택	입력 안 한 후 초과시간 지났을 때 반응 확인	초기화면으로 돌아가야 함
RUC 02 음료 종류 및 수량 선택	숫자가 아닌 문자열을 넣었을 때 반응 확인	문자열이 들어왔으니 재입력 요청
RUC 02 음료 종류 및 수량 선택	수량을 0 및 음수로 입력했을 때 반응 확인	0과 음수 수량은 유효하지 않으므로 재입력 요청,
RUC 02 음료 종류 및 수량 선택	구매하고 싶은 수량은 자리수 제한이 있는지	큰 값이 입력될 경우 최대값 이하로 입력해달라고 메세지 출력, 최대값이 몇으로 되어 있는가
RUC 02 음료 종류 및 수량 선택	자판기에서 판매 가능한 전체 음료 아이템 목록을 출력한다.	1부터 20까지의 메뉴 display
RUC 03 카드 정보 입력과 확인	음료 가격이 알맞게 계산되는지 확인	음료 가격 * 수량으로 문제 없음
RUC 03 카드 정보 입력과 확인	유효한 카드정보와 유효하지 않은 카드정보 입력 시 결과 확인	유효한 카드 정보 입력시 결제가 완료되며, 유효하지 않은 카드 정보는 다시 입력 요청
RUC 03 카드 정보 입력과 확인	카드번호에 공백, 특수문자가 포함 된 경우	잘못된 형식으로 간주되어 재입력 요청
RUC 04 음료 판매 처리	결제 후 판매 목록에 정상적으로 Sale 객체가 추가되는지 확인	판매 기록이 생성되고 저장됨
RUC 04 음료 판매 처리(현재 자판기)	결제 완료 후 Enter 누를 경우와 다른 key를 눌렀을 때 확인	Enter를 입력하면 메인 화면으로 돌아감
RUC 04 음료 판매 처리(현재 자판기)	음료 결제 후 자판기에서 해당 수량만큼 차감되는지 확인	해당 수량만큼 차감 됨
RUC 03 다른 자판기 조회 및 위치 안내	현재 자판기 및 다른 자판기의 재고 상태를 확인하고 본인의 재고가 충분할 경우 판매한다.	재고가 충분하지 않다면 가장 가까운 다른 판매기의 위치를 출력 및 선결제 진행
RUC 05 다른 자판기 조회 및 위치 안내	본인의 재고가 충분하지 않을 경우 다른 자판기의 위치를 알려주거나 not available을 출력한다.	다른 자판기의 위치나 not available 메세지 출력
RUC 05 다른 자판기 조회 및 위치 안내	카드 정보 잘못 입력 시, 초기화면으로 돌아가는건 아닌지 확인	카드 정보 잘못 입력 시, 재입력 요청
RUC 05 다른 자판기 조회 및 위치 안내	다른 자판기에도 재고가 없으면 어떤 메세지 나오는지	재고가 없어서 판매할 수 없다는 메세지 출력
RUC 05 다른 자판기 조회 및 위치 안내	재고가 있는 다른 자판기 위치가 제대로 출력되는지	(3, 4) 위치의 자판기에서 판매가능하다는 식의 메세지 출력
RUC 05 다른 자판기 조회 및 위치 안내	인증코드가 정상적으로 5자리로 생성되는지 확인	영숫자 5자리 코드 생성
RUC 05 다른 자판기 조회 및 위치 안내	구매 가능한 자판기 위치와 인증 코드 출력되는지	위치 (3, 4) 인증코드 73GJr과 같은 형식으로 출력
RUC 06 선결제 구입	특정 DVM 대상의 선결제를 처리하고 인증 코드를 반환한다.	선결제 금액을 Sale 객체에 판매 아이템 업데이트 동시에 총 판매 금액을 해당 금액 만큼 증가시켜준다.
RUC 07 선결제 음료 수령	유효한 인증코드와 유효하지 않은 코드 입력 시 결과 확인	유효한 인증코드만 허가되어 음료 수령 가능
RUC 07 선결제 음료 수령	음료 수령 시, 어떤 자판기에서 수량이 빠져나가는지 확인	선결제한 자판기에서 음료 수량이 차감 됨
RUC 07 선결제 음료 수령	음료 제공 후 메인 화면으로 돌아갈 때 Enter가 아닌 다른 키가 입력했을 경우 확인	Enter를 입력하라는 메세지 출력
RUC 07 선결제 음료 수령	인증코드를 입력받아 선결제된 음료를 수령 처리	sales 내에 일치하는 certCode가 있는 경우 수령 처리
RUC 07 선결제 음료 수령	인증 코드를 한 번만 사용할 수 있도록 사용 여부 관리를 한다.	사용 여부를 boolean으로 잘 저장하고 있어야 함
RUC 07 선결제 음료 수령	입력된 인증코드가 일치하고, 아직 사용되지 않았다면 사용 처리를 한다.	사용 처리 후 isUsed true로 업데이트

TestNum	pass/fail	비고
BF-01	pass	
BF-02	pass	
BF-03	pass	
BF-04	pass	
BF-05	pass	
BF-06	pass	
BF-07	pass	
BF-08	pass	
BF-09	pass	
BF-10	pass	
BF-11	pass	재입력요청은 아닌 초기화면으로 돌아감
BF-12	pass	
BF-13	pass	
BF-14	pass	
BF-15	pass	
BF-16	pass	
BF-17	pass	
BF-18	pass	
BF-19	pass	
BF-20	pass	
BF-21	pass	
BF-22	pass	
BF-23	pass	
BF-24	pass	
BF-25	pass	
BF-26	pass	
BF-27	pass	
BF-28	pass	

Pass rate = 100%

- 2nd Cycle에서 Fail 되었던 모든 Brute Force Test Case에 대해 100% Pass

Static Code Analysis Overview
Sonar Cloud Result



- Coverage 부족 문제
  - Application 패키지 영역 63.4% 커버리지 확보
  - 주요 로직에 대한 단위 테스트 추가 필요
  - 기존에 선정한 Unit Test 목적

  (Application Layer에 존재하는 도메인 로직과 비즈니스 로직 화면 출력 및 외부 시스템 연결은 제외)과 같이
  common, infra, presentation 패키지 영역은 커버되지 않음

- Security Hotspot 1건
  - DVMServer 혹은 Prepayment 로직의 TCP 연결에서 보안 문제가 발생할 것으로 예상. 추가 검토가 필요함.

- Duplications
  - 0% 로 코드 중복 없이 모듈화가 잘 이루어짐.

Static Code Analysis Overview
Sonar Cloud Result

```
TEST(PrepaymentHandlerTest, IsvalidCode) {
    FakeBeverageRepository fakeRepo;
    Inventory inventory(&fakeRepo);
    FakeCertificationRepository certRepo;

    PrepaymentHandler handler( broadCast: nullptr, &certRepo, &inventory);
    EXPECT_TRUE(handler.IsvalidCode( "aA1bB2cC"));
    EXPECT_FALSE(handler.IsvalidCode( "short"));                // 길이 부족
    EXPECT_FALSE(handler.IsvalidCode( "toolongcode123"));       // 길이 초과
    EXPECT_FALSE(handler.IsvalidCode( "invalid!@#"));           // 특수문자 포함
    EXPECT_FALSE(handler.IsvalidCode( " "));                     // 공백 포함
}

TEST(PrepaymentHandlerTest, RollBack_Success) {

  FakeBeverageRepository fakeRepo;
  Inventory inventory(&fakeRepo);
  FakeCertificationRepository certRepo;

  PrepaymentHandler handler( broadCast: nullptr, &certRepo, &inventory);

  int initialStock = inventory.getBeverage( code: 1)->getStock();
  std::string certCode = "TEST123";
  int itemCode = 1;
  int qty = 2;


  handler.handlePrepaymentRequest( certCode, itemCode, qty);
  bool result = handler.rollBackPrepaymentRequest( certCode, itemCode, qty);
  EXPECT_TRUE(result);
  EXPECT_TRUE(certRepo.deleteCalled);
  EXPECT_EQ(certRepo.savedCode, "");
  EXPECT_EQ(inventory.getBeverage(itemCode)->getStock(), qty);
```

```
// MAX STOCK 보다 크게 증가시켰을 시
TEST(InventoryTest, IncreaseStock_MAX) {
  FakeBeverageRepository fakeRepo;
  Inventory inventory(&fakeRepo);
  Beverage* testBeverage = inventory.getBeverage(1);
  int testBeverageQty = testBeverage->getStock();
  inventory.increaseStock(1, testBeverage->getMaxStock() + 1);

  EXPECT_EQ(testBeverage->getStock(), testBeverageQty);
}

// MAX STOCK 보다 작게 증가시켰을 때
TEST(InventoryTest, IncreaseStock_Success) {
  FakeBeverageRepository fakeRepo;
  Inventory inventory(&fakeRepo);
  Beverage* testBeverage = inventory.getBeverage(1);
  int testBeverageQty = testBeverage->getStock();
  inventory.increaseStock(1, 10);

  EXPECT_EQ(testBeverage->getStock(), testBeverageQty + 10);
}
```

- Coverage 부족 문제
  - 기존 UT 에서 커버하지 못한 테스트 추가
  - 이후 계속 추가해야 함

Static Code Analysis Overview
Sonar Cloud Result

```
string DVMServer::makeResponsePrepay(json jsonReq) {

  // 필수 필드 검사
  if (!jsonReq.contains( key: "msg_content") || !jsonReq["msg_content"].is_object()) {
    return R"({"error": "Missing or invalid 'msg_content'"})";
  }
  json& content = jsonReq["msg_content"];

  if (!content.contains( key: "item_code") || !content["item_code"].is_string()) {
    return R"({"error": "Missing or invalid 'item_code'"})";
  }
  if (!content.contains( key: "item_num") || !content["item_num"].is_number_integer()) {
    return R"({"error": "Missing or invalid 'item_num'"})";
  }
  if (!content.contains( key: "cert_code") || !content["cert_code"].is_string()) {
    return R"({"error": "Missing or invalid 'cert_code'"})";
  }
  if (!jsonReq.contains( key: "src_id") || !jsonReq["src_id"].is_string()) {
    return R"({"error": "Missing or invalid 'src_id'"})";
  }
```

```
  // 요청 처리 및 응답 생성
  json jsonReq;
  try {
    jsonReq = json::parse( [&] request);
  }catch (const nlohmann::json::parse_error& e) {
    std::cerr << "[DVMServer] JSON 파싱 오류: " << e.what() << std::endl;
    std::string errorResp = R"({"error": "Invalid JSON format"})";
    send(clientFd,  buf: errorResp.c_str(),  len: errorResp.size(),  flags: 0);
    close(clientFd);
    return;
  }

  // msg_type 검사
  if (!jsonReq.contains( key: "msg_type") || !jsonReq["msg_type"].is_string()) {
    std::string errorResp = R"({"error": "Missing or invalid 'msg_type'"})";
    send(clientFd,  buf: errorResp.c_str(),  len: errorResp.size(),  flags: 0);
    close(clientFd);
    return;
  }
```

```
string DVMServer::makeResponseStock(json jsonReq) {
  // 필수 필드 검사
  if (!jsonReq.contains( key: "msg_content") || !jsonReq["msg_content"].is_object()) {
    return R"({"error": "Missing or invalid 'msg_content'"})";
  }
  json& content = jsonReq["msg_content"];

  if (!content.contains( key: "item_code") || !content["item_code"].is_string()) {
    return R"({"error": "Missing or invalid 'item_code'"})";
  }

  if (!jsonReq.contains( key: "src_id") || !jsonReq["src_id"].is_string()) {
    return R"({"error": "Missing or invalid 'src_id'"})";
  }
```

- Security Hotspot Resolution
  - DVMServer 에서 json 요청을 받을 때 json 형식과 유효성에 대한 검사를 추가함.

Act. OOAD – DVM Project Completion

202111308 - 손승우

- OOAD/OOPT 의 프로세스를 따라가면서 Planning, Analysis, Design, Implement까지 해보면서 느낀 점으로 이러한 프로세스의 적용이 복잡하였지만, 막상 사용하고 나니 구현과 테스트, 유지보수에 있어서 상당히 많은 이점을 가진다는 것을 알게 되었다.
- 이론으로만 배웠던 SE, OOAD에 대해서 실전적 경험을 쌓을 수 있는 계기가 되어 즐겁게 프로젝트에 임했던 것 같다.
- 한 학기가 조금 안되는 짧은 시간동안 비교적 작은 프로젝트를 맡게 되었는데, 앞으로 이런 기회를 자주 만들어서, 전공자로써 더욱 체계적인 개발을 할 수 있도록 노력해야 함을 절실히 느꼈다.
- 아쉬웠던 점은, 개발시간이 부족해서 해보고 싶었던 다양한 개념들을 사용하지 못한 것과, 검증팀과 제대로 손발이 맞지 않아서 제대로 검증이 안되었던 점이 아쉽다.
- 또한, 아쉽게도 다른 팀과의 연결에 대해서 제대로 이루어지지 않아서 아쉬웠던 것 같다.

Act. OOAD – DVM Project Completion

202111340 이수민

- 요구사항 문서를 보고, 실제 개발 방법론에 따라 설계하고 수정해가며 초반 설계에 대한 중요성을 많이 느낄 수 있었습니다.
- 이전에 무언가 개발을 하면, 설계 없이 바로 코드부터 짜기 시작해서 제작하는 기간이 늘어날수록 레거시 코드가 점점 늘어나게 되는 경험이 많았는데, 강의를 듣고 해당 내용을 직접 실습해보며 초반 설계가 자세하게 진행될 수록 레거시코드도 줄어들고, 실제 구현 단계에서도 빠르게 구현이 완료됨을 느낄 수 있었습니다.
- 검증단계에서 수정을 진행할 때에도 기존에 설계내용이 있어 수정에도 더 안정적으로 대응할 수 있어 소프트웨어 개발 프로세스와 설계의 중요성을 많이 알고, 실습해보는 경험을 얻을 수 있었습니다

Act. OOAD – DVM Project Completion

202111240 강찬욱

- 방법론을 배우며 적용한 설계와 개발을 진행하면서, 어떻게 하는 것이 객체지향적으로 이상적인가를 끊임없이 고민했던 것 같다. 최적의 방법이 무엇인지 찾으려다 보니 많은 시간이 들었다. 그러나 계속 수업을 듣다 보니 한가지 완벽한 정답은 존재하지 않으며, 수많은 방법 중 현재 상황에 가장 적합한 방법을 선택하는 것이 중요하다는 것을 알게 되었다.
- 개발을 시작해 보니 그동안 몸에 밴 개발 습관과 설계 간의 괴리가 상당히 컸다. 설계를 보고 그대로 구현하려 해도 코드로 표현하기 어려운 부분들이 있었고, 설계 단계에서는 고려하지 못했던 현실과 다른 개념들이 코드에 반영되며 설계를 온전히 따를 수 없는 상황이 종종 발생했다.
- 객체지향은 현실 객체들의 상호작용을 모델로 하여 나온 패러다임인데 그렇게 맞춰서 개발하려다 보니 너무 많고 추적할 수 없는 의존성이 생겼다. 또 프로그래밍에서는 현실에 없는 것들도 클래스로 만들어야 해서 설계 단계에서 커버할 수 없는 것들이 있었고 결국 설계대로 따라갈 수 없는 상황이 많이 발생한 것 같다. 방법론을 적용해본 것이 처음이라 이런 상황을 몰랐었고, 다음에 적용한다면 개발할 때의 괴리도 고려해서 적용해볼 것 같고, 처음부터 다시 설계해보고 싶은 마음도 생겼다.
- 설계를 오래 하는 것이 필요할까? 라는 생각도 했지만 이후 테스트팀에서 결과를 확인하고 코드를 고치려고 할 때 그 문서들이 많이 도움이 됐던 것 같다. 진짜 전문가들의 설계들을 따라서 코드를 작성하면 훨씬 쉬울 것 같다는 느낌도 들었다.
- 평소에도 객체지향프로그래밍에 관심이 많았는데 내가 하고 있던 건 절차지향에 가까웠다 라는 것을 알았고, 이후 개발에서 배웠던 것을 적용해보고 싶다. 너무 재미있었던 수업이었고 아직 궁금한 것이 많지만 객체지향에 대해 많은 생각을 해볼 수 있어서 좋은 경험이었다.