#6 - the 3nd Cycle Presentation

컴퓨터공학부 김진솔
컴퓨터공학부 백현준
컴퓨터공학부 이현수

목차

Category Partition Test - *1 cycle(검증)*

Test Num	P/F	Expected Output	Test Output
CP01	P	유효하지 않음	유효하지 않음
CP02	F	음료 종류 관련 에러 메시지	결제 관련 에러메시지
CP03	F	에러 메시지 출력	미출력
CP04	-	에러 메시지 출력	(음수 재고 입력 불가)
CP05	F	에러 메시지 출력	재고 감소 확인 실패
CP06	-	에러 메시지 출력	(음수 재고 입력 불가)
CP07	F	에러 메시지 출력	선결제 거절 불가
CP08	F	에러 메시지 출력	다른 입력 불가
CP09	F	에러 메시지 출력	10초간 입력 대기 불가
CP10	F	결제 거절 메시지 출력	카드 입력 부분 미구현
CP11	F	결제 거절 메시지 출력	카드 입력 부분 미구현
CP12	-	결제 거절 메시지 출력	카드 금액 0 미만 설정 불가
CP13	-	(Invalid)	(출력 정의 불가)
CP14	F	에러 메시지 호출	선결제 코드 유효
CP15	F	에러 메시지 호출	선결제 코드 처리
CP16	F	에러 메시지 호출	선결제 코드 처리
CP17	F	음료 정상 출고	타 자판기 재고 요청 수행
CP18	F	에러 메시지 호출	(음료 재고 확인 불가)
CP19	F	음료 정상 출고	비정상 출력
CP20	F	음료 정상 출고	결제 거절 및 비정상 출력
CP21	F	음료 정상 출고	(음료 재고 확인 불가)

- **PASS: 4**
- **INVALID: 8**
- **TOTAL: 16**
- **PASS/VALID: 4/9**
- **CPT Pass Percentage = 44.44%**

Category Partition Test - *2 cycle*

Test Num	P/F	Expected Output	Test Output
CP01	P	유효하지 않음.	유효하지 않음.
CP02	F → P	에러메시지 출력 (재고 조회 브로드캐스트 E1)	에러 메시지 출력
CP03	F	에러 메시지 출력	에러메시지 없음
CP04	-	에러 메시지 출력	음료 재고를 0 미만으로 변경할 방법이 없음
CP05	F → P	에러 메시지 호출	에러 메시지 호출
CP06	-	에러 메시지 출력	음료 재고를 0 미만으로 변경할 방법이 없음
CP07	F → P	에러 메시지 출력	에러 메시지 출력
CP08	F	에러 메시지 출력	선결제 진행됨
CP09	F	에러 메시지 출력	에러 메시지 미출력
CP10	F → -	결제 거절 메시지 출력	카드 번호 입력 부분 미구현
CP11	F → -	결제 거절 메시지 출력	카드 번호 입력 부분 미구현

Category Partition Test- *2 cycle*

Test Num	P/F	Expected Output	Test Output
CP12	-	결제 거절 메시지 출력	카드 금액을 0 미만으로 설정 불가
CP13	-	정의 불가	정의 불가
CP14	F → P	에러 메시지 호출	에러 메시지 호출
CP15	F → P	에러 메시지 호출	에러 메시지 호출
CP16	F → P	에러 메시지 호출	에러 메시지 호출
CP17	F → P	음료 정상 출고	음료 정상 출고
CP18	F → P	에러 메시지 출력	에러 메시지 출력
CP19	F → P	가까운 거리 안내 및 음료 출고	가까운 거리 안내 및 음료 출고
CP20	F → P	음료 정상 출고	음료 정상 출고
CP21	F → P	T2 자판기에서 사이다 구매 성공	T2 자판기에서 사이다 구매 성공

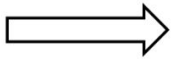
Category Partition Test - *2 cycle*

Test Num	P/F	Expected Output	Test Output
CP12	-	결제 거절 메시지 출력	카드 금액을 0 미만으로 설정 불가
CP13	-	정의 불가	정의 불가
CP14	F → P	에러 메시지 호출	에러 메시지 호출
CP15	F → P	에러 메시지 호출	에러 메시지 호출
CP16	F → P	에러 메시지 호출	에러 메시지 호출
CP17	F → P	음료 정상 출고	음료 정상 출고
CP18	F → P	에러 메시지 출력	에러 메시지 출력
CP19	F → P	가까운 거리 안내 및 음료 출고	가까운 거리 안내 및 음료 출고
CP20	F → P	음료 정상 출고	음료 정상 출고
CP21	F → P	T2 자판기에서 사이다 구매 성공	T2 자판기에서 사이다 구매 성공

Category Partition Test Resolution

- 1st Cycle

- Total: 21
- Pass: 1
- Invalid: 4
- PASS/VALID: 1/17
- Pass Percentage = **5.88%**



- 2nd Cycle

- Total: 21
- Pass: 12
- Invalid: 6
- PASS/VALID: 12/15
- Pass Percentage = **80%**

Category Partition Test - *2 cycle*

Test Num	P/F	Expected Output	Test Output
CP01	P	유효하지 않음.	유효하지 않음.
CP02	F → P	에러메시지 출력 (재고 조회 브로드캐스트 E1)	에러 메시지 출력
CP03	F	에러 메시지 출력	에러메시지 없음
CP04	-	에러 메시지 출력	음료 재고를 0 미만으로 변경할 방법이 없음
CP05	F → P	에러 메시지 호출	에러 메시지 호출
CP06	-	에러 메시지 출력	음료 재고를 0 미만으로 변경할 방법이 없음
CP07	F → P	에러 메시지 출력	에러 메시지 출력
CP08	F	에러 메시지 출력	선결제 진행됨
CP09	F	에러 메시지 출력	에러 메시지 미출력
CP10	F → -	결제 거절 메시지 출력	카드 번호 입력 부분 미구현
CP11	F → -	결제 거절 메시지 출력	카드 번호 입력 부분 미구현

Category Partition Test Resolution - *3 cycle*

cp -03

| 음료 선택 메뉴에서 30초 이상 대기

- 테스트 순서
    - b. (음료선택 및 결제)
    - 30초 이상 대기
- expected output: 에러메시지 출력

cp -09

| 선결제 승인/거절 선택지에서 10초 대기

- 테스트 순서
    - 자판기 개수 2개
    - 1번 자판기에서 2. (음료 선택 및 결제)
    - 음료: 박카스 (원격 자판기 음료)
    - 현재 자판기 음료 개수: 0개
    - 원격 자판기 음료 개수: 0개 초과
    - 선결제: (입력 없음, 10초 간 대기)
- expected output: 에러 메시지 출력

Category Partition Test Resolution - *3 cycle*

cp -03

cp -09

입력스레드 생성 및 타임아웃 설정

```cpp
// 입력 스레드
std::thread inputThread([&]() {
    std::string line;
    std::getline(std::cin, line);
    {
        std::lock_guard<std::mutex> lock(mtx);
        userInput = line;
        ready = true;
    }
    cv.notify_one();
});

// 타임아웃 대기
std::unique_lock<std::mutex> lock(mtx);
if (cv.wait_for(lock, timeout, [&]{ return ready; })) {
    // 시간 내에 입력 성공
    inputThread.join();

    // 입력값 정리
    const auto first = userInput.find_first_not_of(" \t\n\r");
    if (std::string::npos == first) { return false; } // 공백만 입력 시 false

    const auto last = userInput.find_last_not_of(" \t\n\r");
    userInput = userInput.substr(first, (last - first + 1));

    // Y이면 true, 그 외에는 모두 false 반환
    return (userInput.length() == 1 && std::toupper(userInput[0]) == 'Y');

} else {
    // 타임아웃 발생
    std::cout << "\n[시간 초과] 입력 시간이 초과되었습니다." << std::endl;
    inputThread.detach();
    return false; // 타임아웃 시 false 반환
}
```

Category Partition Test Resolution - *3 cycle*

cp -08

| 선결제 승인/거절 선택지에서 승인 거절 외의 다른 입력

- 테스트 순서
  - 자판기 개수 2개
  - 1번 자판기에서 2. (음료 선택 및 결제)
  - 음료: 박카스 (원격 자판기 음료)
  - 현재 자판기 음료 개수: 0개
  - 원격 자판기 음료 개수: 0개 초과
  - 선결제: ANDKEIN
- expected output: 에러 메시지 출력

```
옵션을 선택하세요:
1. 음료 선택 (구매/다른 자판기 조회)
/home/dev/src • 강조 표시한 항목 포함
선택 (숫자 입력): 1
구매할 음료의 2자리 코드를 입력하세요 (메뉴로 돌아가려면 'c' 또는 'C' 입력): 02
사이다 선택됨. 가격: 1000원. (재고: 3개)
사이다 구매. 가격: 1000원.

결제할 금액: 1000원 입니다.
카드를 삽입해주세요. (시뮬레이션: 실제 카드 처리 없음)
결제를 계속 진행하시겠습니까? (Y/N): u
잘못된 입력입니다. Y 또는 N을 입력해주세요.
결제를 계속 진행하시겠습니까? (Y/N): a
잘못된 입력입니다. Y 또는 N을 입력해주세요.
결제를 계속 진행하시겠습니까? (Y/N): d
잘못된 입력입니다. Y 또는 N을 입력해주세요.
결제를 계속 진행하시겠습니까? (Y/N):
잘못된 입력입니다. Y 또는 N을 입력해주세요.
결제를 계속 진행하시겠습니까? (Y/N): vvvvv
잘못된 입력입니다. Y 또는 N을 입력해주세요.
결제를 계속 진행하시겠습니까? (Y/N): y
결제 승인 중입니다. 잠시만 기다려주세요...
[결제 성공] 결제가 성공적으로 완료되었습니다!

[사이다] 음료가 배출되고 있습니다. 잠시만 기다려주세요...
[사이다] 음료가 나왔습니다. 이용해주셔서 감사합니다!
거래가 완료되었습니다. 감사합니다.
```

Brute Force Test - *1 cycle*

Test	Test Num	Test Result	Detail
Select Menu	1-1	FAIL	프로그램 종료
	1-2	FAIL	미출력
	1-3	PASS	에러메시지 출력
	1-4	FAIL	입력값 유효
Select Drink	2-1	FAIL	미출력
	2-2	FAIL	결제 거절 메시지 출력
	2-3	PASS	에러 메시지 출력
	2-4	FAIL	미출력
	2-5	FAIL	문자가 깨져 확인 불가
Prepayment	3-1	FAIL	문자가 깨져 확인 불가
	3-2	FAIL	미구현
	3-3	FAIL	미출력
	3-4	FAIL	문자가 깨져 확인 불가
	3-5	FAIL	다수의 인증코드
	3-6	FAIL	문자가 깨져 확인 불가
	3-7	FAIL	문자가 깨져 확인 불가
	3-8	FAIL	문자가 깨져 확인 불가
Provide Drink	4-1	FAIL	비정상 출력
Card System	5-1	FAIL	미구현
	5-2	FAIL	미구현
	5-3	FAIL	미구현
DVMs location	6-1	FAIL	문자가 깨져 확인 불가
	6-2	FAIL	위치 좌표 확인 불가로 인한 확인 실패

PASS: 2

INVALID: 0

TOTAL: 23

PASS/VALID: 2/23

BFT Pass Percentage = 8.69%

Brute Force Test Resolution - *2 cycle*

Test	Test Num	Description	Test Result	Expected Output	Test Output
Select Menu	1-1	한글, 영문, 특수문자 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	1-2	공란 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	1-3	음수 입력	PASS	에러 메시지 출력	에러메시지 출력
	1-4	여러 메뉴 동시 입력	INVALID	-	-
Select Drink	2-1	음료 선택하지 않고 기다리기	INVALID	-	-
	2-2	여러 음료 동시에 선택	INVALID	-	-
	2-3	숫자, 영문, 특수문자 입력	PASS	에러 메시지 출력	에러 메시지 출력
	2-4	공란 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	2-5	현재 자판기 재고 확인	INVALID	-	-

Brute Force Test Resolution - *2 cycle*

Test	Test Num	Description	Test Result	Expected Output	Test Output
Prepayment	3-1	유효하지 않은 인증코드 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	3-2	사용한 인증코드 재입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	3-3	인증코드 공란 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	3-4	인증코드 특수문자 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	3-1	인증코드 중복생성 확인	FAIL → PASS	하나의 인증코드	하나의 인증코드
	3-2	인증코드 한글로 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	3-3	안내되지 않은 VM에 인증코드 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력
	3-4	서로 다른 DVM에서 같은 코드 동시 입력	FAIL → PASS	에러 메시지 출력	에러 메시지 출력

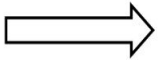
Brute Force Test Resolution - *2 cycle*

Test	Test Num	Description	Test Result	Expected Output	Test Output
Provide Drink	4-1	정확한 음료 제공 확인	FAIL → PASS	음료 제공 확인 출력	음료 제공 확인 출력
Card System	5-1	유효하지 않은 카드 정보 입력	INVALID	-	-
	5-2	카드 정보 공란 입력	INVALID	-	-
	5-3	카드 정보 영문, 한글, 특수문자 입력	INVALID	-	-
DVMs location	6-1	DVMs 위치 출력 확인	FAIL → PASS	다른 DVM의 위치좌표 출력	다른 DVM의 위치좌표 출력
	6-2	다른 DVM의 전원 off 후 위치 출력 확인	FAIL → PASS	에러 메시지 출력	에러 메시지 출력

Brute Force Test Resolution

- 1st Cycle

- Total: 23
- Pass: 2
- Invalid: 0
- PASS/VALID: 2/23
- Pass Percentage = **8.69%**



- 2nd Cycle

- Total: 23
- Pass: 16
- Invalid: 7
- PASS/VALID: 16/16
- Pass Percentage = **100%**

Static Analysi(High) - *2 cycle*

Type	Count	Rule
Global 변수 관련	4	Global variables should be const.
Cognitive Complexity 관련	2	Refactor this function to reduce its Cognitive Complexity from 42 to the 25 allowed. Refactor this function to reduce its Cognitive Complexity from 25 to the 25 allowed.
Cognitive Complexity 관련	2	Refactor this code to not nest more than 3 if|for|do|while|switch statements.
std::function 성능 최적화	2	Replace this "std::function" with a template parameter.
멀티 락 처리 개선	1	Use C++ facilities as "std::scoped_lock" to acquire multiple mutexes.

Static Analysis(High) Resolution - *3 cycle*

Global 변수 관련	4	Global variables should be const.

main.cpp

자판기 설정을 위한 전역변수
➔ 구조체로 변경

```
// 자판기 설정 구조체
struct VendingMachineConfig {
    std::string id = "T6";
    int x = 50;
    int y = 50;
    unsigned short port = 12350;
};
```

Static Analysis(High) Resolution - *3 cycle*

Cognitive Complexity 관련	2	Refactor this function to reduce its Cognitive Complexity from 42 to the 25 allowed. Refactor this function to reduce its Cognitive Complexity from 25 to the 25 allowed.

main.cpp

명령행 인자를 파싱하는 로직이 너무 복잡함

➔ 별도의 함수로 분리

```
void setupGeneralInventory(const std::string& currentVmId, persistence::InventoryRepository& inventoryRepo) {
    std::cout << "정보: " << currentVmId << " 자판기의 일반 재고를 설정합니다." << std::endl;
    if (currentVmId == "T6") { // 우리 조 자판기 (T6)
        inventoryRepo.addOrUpdateStock(domain::Inventory("01", 0));  // 콜라 (선결제 테스트용으로 재고 없음)
        inventoryRepo.addOrUpdateStock(domain::Inventory("02", 5));  // 사이다 (로컬 구매 가능)
        inventoryRepo.addOrUpdateStock(domain::Inventory("03", 10)); // 녹차
        inventoryRepo.addOrUpdateStock(domain::Inventory("06", 0));  // 탄산수 (선결제 테스트용으로 재고 없음)
        inventoryRepo.addOrUpdateStock(domain::Inventory("08", 7));  // 캔커피
        inventoryRepo.addOrUpdateStock(domain::Inventory("15", 3));  // 오렌지주스
        inventoryRepo.addOrUpdateStock(domain::Inventory("20", 4));  // 카페라떼
        std::cout << "  T6: 콜라(0), 사이다(5), 녹차(10), 탄산수(0), 캔커피(7), 오렌지주스(3), 카페라떼(4) 설정됨." << std::endl;
    } else if (currentVmId == "T1") { // T6가 선결제할 대상 자판기 예시
        inventoryRepo.addOrUpdateStock(domain::Inventory("01", 10)); // 콜라 (T6가 선결제 가능)
        inventoryRepo.addOrUpdateStock(domain::Inventory("02", 0));
        inventoryRepo.addOrUpdateStock(domain::Inventory("04", 8));  // 홍차
        inventoryRepo.addOrUpdateStock(domain::Inventory("09", 15)); // 물
        inventoryRepo.addOrUpdateStock(domain::Inventory("10", 3));
        inventoryRepo.addOrUpdateStock(domain::Inventory("11", 6));
        inventoryRepo.addOrUpdateStock(domain::Inventory("12", 9));
        std::cout << "  T1: 콜라(10), 사이다(0), 홍차(8), 물(15), 에너지드링크(3), 유자차(6), 식혜(9) 설정됨." << std::endl;
    } else if (currentVmId == "T2") {
        inventoryRepo.addOrUpdateStock(domain::Inventory("02", 12)); // 사이다
        inventoryRepo.addOrUpdateStock(domain::Inventory("06", 10)); // 탄산수 (T6가 선결제 가능)
        inventoryRepo.addOrUpdateStock(domain::Inventory("13", 5));
        inventoryRepo.addOrUpdateStock(domain::Inventory("17", 6));
        std::cout << "  T2: 사이다(12), 탄산수(10), 아이스티(5), 이온음료(6) 설정됨." << std::endl;
    }
    // 다른 자판기들에 대해서도 필요에 따라 다양한 재고 설정
    else {
    }
}
```

Static Analysis(High) Resolution - *3 cycle*

std::function 성능 최적화	2	Replace this "std::function" with a template parameter.

network/
PaymentCallbackReceiver.hpp

콜백 함수가 간단한 동작을 수행함
➔ 템플릿 함수로 변경해 컴파일러가
콜백 함수를 최적화 할 수 있게 함

```cpp
#pragma once

#include <functional>
#include <thread>
#include <chrono>
#include <random>
namespace network {
class PaymentCallbackReceiver {
public:
    using Callback = std::function<void(bool success)>;

    template <typename T_Callback> // 템플릿 파라미터 사용
    void simulatePrepayment(const T_Callback& cb, int delay_seconds = 3) const {
        std::this_thread::sleep_for(std::chrono::seconds(delay_seconds));
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_real_distribution<> dist(0.0, 1.0);
        bool success = dist(gen) < 0.99;
        cb(success);
    }
};

} // namespace network
```

Static Analysis(High) Resolution - *3 cycle*

Cognitive Complexity 관련	2	Refactor this code to not nest more than 3 if|for|do|while|switch statements.

presentation/
UserInterface.cpp

if/for문이 4단계로 중첩 되있다
➔ 중첩된 로직을 별도의 헬퍼함수로 분리

Static Analysis(High) Resolution - *3 cycle*

Cognitive Complexity 관련	2	Refactor this code to not nest more than 3 if|for|do|while|switch statements.

service/
User Process Controller.cpp

처리로직이 깊게 중첩돼있음
➔ 중첩된 로직을 별도의
헬퍼함수로 분리

```
void UserProcessController::handleStockResponseTimeout() {
    std::lock_guard<std::mutex> lock(mtx_);

    if (currentState_ != ControllerState::AWAITING_STOCK_RESPONSES) {
        return;
    }

    if (!availableOtherVmsForDrink_.empty()) {
        currentState_ = ControllerState::DISPLAYING_OTHER_VM_OPTIONS;
    } else {
        last_error_info_ = errorService_.processOccurredError(ErrorType::RESPONSE_TIMEOUT_FRO
        currentState_ = ControllerState::HANDLING_ERROR;
    }
}
```

Static Analysis(Medium) - *2 cycle*

(C++) Generic exceptions should not be cau...	28
(C++) Sections of code should not be comm...	12
(C++) Pass by reference to const should be u...	11
(C++) Transparent function objects should be ...	8
(C++) "std::format" should be used instead of ...	6
(C++) Member functions that don't mutate the...	6
(C++) The underlying value of an enum shoul...	4
(C++) Unused function parameters should be ...	3
(C++) Emplacement should be preferred whe...	2
(C++) Member data should be initialized in-cla...	2
(C++) Nested blocks of code should not be lef...	2
(C++) Structured binding should be used	2
(C++) Try-catch blocks should not be nested	2
(C++) "auto" should be used to avoid repetitio...	1
(C++) "std::jthread" should be used instead of ...	1
(C++) Functions should not have too many par...	1
(C++) Generic exceptions should never be thr...	1
(C++) Structures should not have too many fiel...	1
(C++) Unused assignments should be removed	1

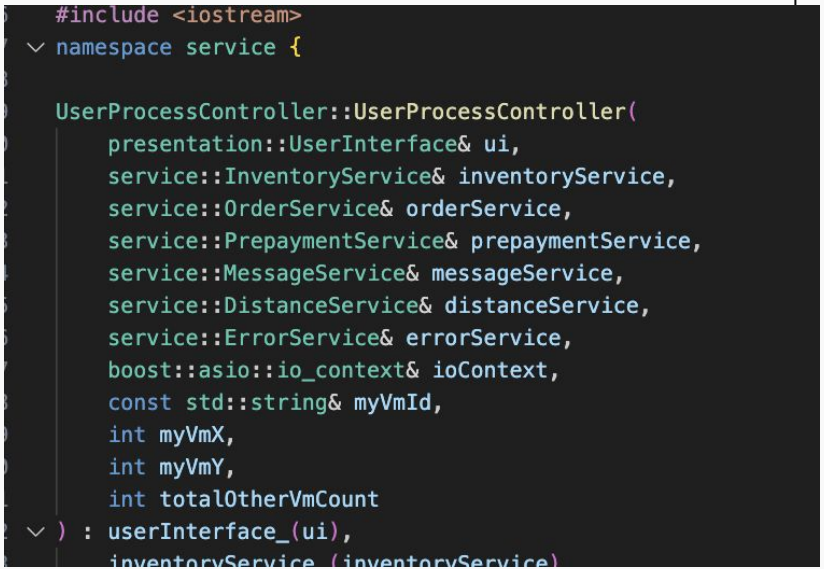
주요 이슈

- 예외 처리 미흡, generic exception catch 사용 과다
- 주석 코드 남용, 중첩 블록, auto 미사용 등으로 유지보수성 저하
- 현대적 C++ 기능 활용 부족
- const 미지정 멤버 함수, 미사용 파라미터, enum 타입 정의 누락, 과다 파라미터 사용

Static Analysis(Medium) Resolution - *3 cycle*

Pass expensive to copy object by reference to const

객체를 전달할 때, pass-by-value 하여 비효율적임
➔ 클래스의 생성자에서 파라미터 타입 수정



Static Analysis(Medium) Resolution - *3 cycle*

Remove the commented out code

여러 파일에 걸쳐 현재 사용되지 않고 주석으로만 남아있는 코드들이 있음

➔ 주석 블록 삭제

```
*/
void OrderService::processOrderApproval(domain::Order& order, bool isPrepayment) {
    // UC5 - Typical Course 2: Order.Paymentstatus 를 APPROVED 로 설정
    try {
        order.setPayStatus("APPROVED");

        if (isPrepayment) {
            // 선결제인 경우: 인증 코드 생성 및 주문에 설정 (UC12의 일부)
            std::string authCode = prepaymentService_.generateAuthCodeString(); // PFR R4.2
            order.setCertCode(authCode);
            // 인증 코드 포함하여 주문 정보 업데이트
            orderRepository_.save(order);


            // UC5 -> UC16 (재고 확보 요청)은 UserProcessController가 MessageService를 통해 진행.
        } else {
            // 일반 구매인 경우: 주문 정보 업데이트 (주로 상태 변경) 후 재고 차감
            orderRepository_.save(order);
            // UC7 - Typical Course 2: 일반결제인 경우 재고를 1 감소
            inventoryService_.decreaseStock(order.getDrinkCode());
        }
        // UC5 - Typical Course 3: 결제 성공 메시지 표시 및 다음 단계 이동은 UserProcessController가 담당.
    } catch (const std::exception& e) {
        order.setPayStatus("ERROR_POST_APPROVAL"); // 오류 발생 시 주문 상태 변경
        try {
            orderRepository_.save(order);
        } catch (...) { }
        errorService_.processOccurredError(ErrorType::UNEXPECTED_SYSTEM_ERROR, "결제 승인 후 내부 처리 오류: " + std::string(e.wha
    }
}
```

Static Analysis(Medium) Resolution - *3 cycle*

Error Handling

Try - Catch 구문에서 너무 포괄
인 예외를 잡아 오류의 원인 명확
파악 할 수 없음
➔ 구체적인 예외를 잡도록 설정

```
*/
void OrderService::processOrderApproval(domain::Order& order, bool isPrepayment) {
    // UC5 - Typical Course 2: Order.Paymentstatus 를 APPROVED 로 설정
    try {
        order.setPayStatus("APPROVED");

        if (isPrepayment) {
            // 선결제인 경우: 인증 코드 생성 및 주문에 설정 (UC12의 일부)
            std::string authCode = prepaymentService_.generateAuthCodeString(); // PFR R4.2
            order.setCertCode(authCode);
            // 인증 코드 포함하여 주문 정보 업데이트
            orderRepository_.save(order);


            // UC5 -> UC16 (재고 확보 요청)은 UserProcessController가 MessageService를 통해 진행.
        } else {
            // 일반 구매인 경우: 주문 정보 업데이트 (주로 상태 변경) 후 재고 차감
            orderRepository_.save(order);
            // UC7 - Typical Course 2: 일반결제인 경우 재고를 1 감소
            inventoryService_.decreaseStock(order.getDrinkCode());
        }
        // UC5 - Typical Course 3: 결제 성공 메시지 표시 및 다음 단계 이동은 UserProcessController가 담당.
    } catch (const std::exception& e) {
        order.setPayStatus("ERROR_POST_APPROVAL"); // 오류 발생 시 주문 상태 변경
        try {
            orderRepository_.save(order);
        } catch (...) { }
        errorService_.processOccurredError(ErrorType::UNEXPECTED_SYSTEM_ERROR, "결제 승인 후 내부 처리 오류: " + std::string(e.wha
    }
}
```

Static Analysis(Medium) Resolution - *3 cycle*

Refactor this structure so it has no more than 20 fields
UserProcessController 클래스가 너무 많은 멤버 변수를 가짐

생성자가 구조체로부터
각 서비스 멤버 변수를
초기화하도록
수정

/home/dev/include/service/OrderService.hpp • 수정

```
UserProcessController::UserProcessController(
    service::InventoryService& inventoryService,
    service::OrderService& orderService,
    service::PrepaymentService& prepaymentService,
    service::MessageService& messageService,
    service::DistanceService& distanceService,
    service::ErrorService& errorService,
    boost::asio::io_context& ioContext,
    const std::string& myVmId,
    int myVmX,
    int myVmY,
    int totalOtherVmCount
) : userInterface_(ui),
    inventoryService_(inventoryService),
    orderService_(orderService),
    prepaymentService_(prepaymentService),
    messageService_(messageService),
    distanceService_(distanceService),
    errorService_(errorService),
    ioContext_(ioContext),
    myVendingMachineId_(myVmId),
    myVendingMachineX_(myVmX),
    myVendingMachineY_(myVmY),
    currentState_(ControllerState::INITIALIZING),
    isCurrentOrderPrepayment_(false),
    total_other_vms_(totalOtherVmCount), // 주입받은 값으로 초기화
    response_timer_(ioContext) {
}

void UserProcessController::run() {
    {
        if (currentState_ == ControllerState::INITIALIZING) {
            initializeSystemAndRegisterMessageHandlers(); // 내부에서 콜백 등록
            userInterface_.displayMessage("자판기 시스템을 시작합니다. 현재 자판기 ID: " + myVendingMachineId_);
            changeState(ControllerState::SYSTEM_READY);
        }
```

Static Analysis(Medium) Resolution - *3 cycle*

객체를 변경하지 않는 함수에 **const** 붙여 안정성 높임
사용되지 않는 변수 제거**(Dead Code)**
사용되지 않는 함수 파라미터 제거
중첩된 **Try - Catch** 구문제거
모던 **C++** 스타일 적용하기
- 구조적 바인딩**(Structured Binding)** 사용
- **auto** 키워드로 타입 추론 활성화

vector에 원소를 추가할 때, **push_back** 대신 **emplace_back**을 사용 ➔
임시 객체 생성 및 복사 과정 생략

Coverage - *2 cycle*

커버리지
0%

Coverage - *2 cycle*

File	Lines			Functions	
domain/prepaymentCode.cpp		0.0%	0 / 8	0.0%	0 / 1
network/MessageReceiver.cpp		92.9%	39 / 42	100.0%	11 / 11
network/MessageSender.cpp		81.2%	26 / 32	100.0%	3 / 3
network/MessageSerializer.cpp		100.0%	24 / 24	100.0%	2 / 2
network/PaymentCallbackReceiver.cpp		100.0%	7 / 7	100.0%	1 / 1
persistence/DrinkRepository.cpp		100.0%	29 / 29	100.0%	2 / 2
persistence/inventoryRepository.cpp		39.0%	16 / 41	66.7%	4 / 6
persistence/OrderRepository.cpp		26.5%	9 / 34	33.3%	2 / 6
persistence/OvmAddressRepository.cpp		0.0%	0 / 16	0.0%	0 / 3
persistence/prepayCodeRepository.cpp		90.5%	19 / 21	100.0%	3 / 3
presentation/UserInterface.cpp		22.6%	33 / 146	30.0%	6 / 20
service/DistanceService.cpp		48.6%	17 / 35	66.7%	2 / 3
service/ErrorService.cpp		38.1%	24 / 63	100.0%	3 / 3
service/InventoryService.cpp		59.3%	32 / 54	80.0%	4 / 5
service/MessageService.cpp		59.3%	51 / 86	66.7%	8 / 12
service/OrderService.cpp		58.0%	29 / 50	80.0%	4 / 5
service/PrepaymentService.cpp		84.1%	58 / 69	100.0%	8 / 8
service/UserProcessController.cpp		2.9%	17 / 585	2.2%	1 / 45

Generated by: GCOVR (Version 8.3)

System Test Case

Use Case 1 음료목록 조회 및 표시

```
// 테스트 1: 음료 목록 표시 기능 테스트
TEST(UC02Test, DisplayDrinkMenu) {
    std::cout << "=== UC02 테스트: 음료 목록 표시 기능 ===" << std::endl;

    // Repository 및 Service 초기화
    persistence::DrinkRepository drinkRepo;
    persistence::InventoryRepository inventoryRepo;
    service::ErrorService errorService;
    service::InventoryService inventoryService(inventoryRepo, drinkRepo, errorService);
    presentation::UserInterface ui;

    // 전체 음료 목록 가져오기 (main.cpp의 displayMainMenu와 동일)
    std::vector<Drink> allDrinks = inventoryService.getAllDrinkTypes();

    std::cout << "전체 음료 수: " << allDrinks.size() << "종" << std::endl;

    // 음료 목록이 올바르게 로드되었는지 확인
    EXPECT_GT(allDrinks.size(), 0); // 최소 1개 이상의 음료

    // 주요 음료들이 포함되어 있는지 확인
    bool hasBasicDrinks = false;
    for (const auto& drink : allDrinks) {
        if (drink.getDrinkCode() == "01" || drink.getDrinkCode() == "02" || drink.getDrinkCode() == "03") {
            hasBasicDrinks = true;
            EXPECT_FALSE(drink.getName().empty());
            EXPECT_GT(drink.getPrice(), 0);
            std::cout << "음료: " << drink.getDrinkCode() << " - " << drink.getName()
                      << " (" << drink.getPrice() << "원)" << std::endl;
        }
    }
    EXPECT_TRUE(hasBasicDrinks);

    // UI 표시 기능 테스트 (실제 출력은 하지 않음)
    std::vector<Inventory> emptyInventory; // 빈 재고 정보
    ui.displayDrinkList(allDrinks, emptyInventory);

    std::cout << "✓ 테스트 1 완료: 음료 목록 표시 기능 정상" << std::endl;
}
```

```
전체 음료 수: 20종
음료: 01 - 콜라 (1000원)
음료: 02 - 사이다 (1000원)
음료: 03 - 녹차 (1000원)

--- 음료 목록 (총 20종) ---
+----------+--------------------+----------+
| 음료코드 | 음료명             | 가격     |
+----------+--------------------+----------+
| 01       | 콜라               |    1000원 |
| 02       | 사이다             |    1000원 |
| 03       | 녹차               |    1000원 |
| 04       | 홍차               |    1000원 |
| 05       | 밀크티             |    1000원 |
| 06       | 탄산수             |    1000원 |
| 07       | 보리차             |    1000원 |
| 08       | 캔커피             |    1000원 |
| 09       | 물                 |    1000원 |
| 10       | 에너지드링크       |    1000원 |
| 11       | 유자차             |    1000원 |
| 12       | 식혜               |    1000원 |
| 13       | 아이스티           |    1000원 |
| 14       | 딸기주스           |    1000원 |
| 15       | 오렌지주스         |    1000원 |
| 16       | 포도주스           |    1000원 |
| 17       | 이온음료           |    1000원 |
| 18       | 아메리카노         |    1000원 |
| 19       | 핫초코             |    1000원 |
| 20       | 카페라떼           |    1000원 |
+----------+--------------------+----------+
✓ 테스트 1 완료: 음료 목록 표시 기능 정상
[       OK ] UC02Test.DisplayDrinkMenu (0 ms)
[----------] 1 test from UC02Test (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
profiling: /Users/baikhyunjun/gitclone/SMA_T6/build/CMakeFiles/runUC01Test.dir/tests/UC01.cpp.gcda: cannot merge previous GCD
ters (92)
profiling: /Users/baikhyunjun/gitclone/SMA_T6/build/CMakeFiles/runUC01Test.dir/tests/UC01.cpp.gcda: cannot merge previous GCD
```

System Test Case

Use Case 2. 사용자음료선택처리

```
TEST(UC02Test, InvalidDrinkCodeInputHandling) {
    std::cout << "=== UC02 테스트: 잘못된 음료 코드 입력 처리 ===" << std::endl;

    // Repository 초기화
    persistence::DrinkRepository drinkRepo;
    std::vector<Drink> allDrinks = drinkRepo.findAll();

    std::cout << "잘못된 입력 처리 테스트" << std::endl;

    // 잘못된 음료 코드들 (UserInterface::selectDrink에서 거부될 입력들)
    std::vector<std::string> invalidCodes = {
        "1",    // 1자리
        "ABC",  // 알파벳
        "99",   // 존재하지 않는 코드
        "00",   // 잘못된 형식
        "123",  // 3자리
        "",     // 빈 문자열
        " 01 "  // 공백 포함
    };

    for (const std::string& code : invalidCodes) {
        // UserInterface::selectDrink의 검증 로직
        bool isValidFormat = (code.length() == 2 &&
                              std::all_of(code.begin(), code.end(), ::isdigit));

        // 음료 목록에 존재하는지 확인
        bool existsInList = false;
        if (isValidFormat) {
            for (const auto& drink : allDrinks) {
                if (drink.getDrinkCode() == code) {
                    existsInList = true;
                    break;
                }
            }
        }

        bool shouldBeRejected = !isValidFormat || !existsInList;
        EXPECT_TRUE(shouldBeRejected);
```

```
d307e)
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC02Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from UC02Test
[ RUN      ] UC02Test.InvalidDrinkCodeInputHandling
=== UC02 테스트: 잘못된 음료 코드 입력 처리 ===
잘못된 입력 처리 테스트
✗ 거부된 입력: '1' - 형식 NG, 존재하지 않음
✗ 거부된 입력: 'ABC' - 형식 NG, 존재하지 않음
✗ 거부된 입력: '99' - 형식 OK, 존재하지 않음
✗ 거부된 입력: '00' - 형식 OK, 존재하지 않음
✗ 거부된 입력: '123' - 형식 NG, 존재하지 않음
✗ 거부된 입력: '' - 형식 NG, 존재하지 않음
✗ 거부된 입력: ' 01 ' - 형식 NG, 존재하지 않음
✓ UC02.3 완료: 잘못된 음료 코드 입력 처리
[       OK ] UC02Test.InvalidDrinkCodeInputHandling (0 ms)
[ RUN      ] UC02Test.DrinkSelectionCompletion
=== UC02 테스트: 음료 선택 완료 및 음료 정보 조회 ===
UC02 결과: 사용자가 선택한 음료 코드 = 01
UC02 완료 - 선택된 음료 정보:
  코드: 01
  이름: 콜라
  가격: 1000원
→ 다음 단계: UC03 재고 확인으로 진행
✓ UC02.5 완료: 음료 선택 과정 완전 종료, UC03으로 전달
[       OK ] UC02Test.DrinkSelectionCompletion (0 ms)
[----------] 2 tests from UC02Test (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

System Test Case

Use Case 3. 현재자판기재고확인

```
TEST(UC03Test, CheckStockForAvailableDrink) {
    std::cout << "=== UC03 테스트: 재고 있는 음료 재고 확인 ===" << std::endl;

    // Repository 및 Service 초기화
    persistence::InventoryRepository inventoryRepo;
    persistence::DrinkRepository drinkRepo;
    service::ErrorService errorService;
    service::InventoryService inventoryService(inventoryRepo, drinkRepo, errorService);

    // T1 자판기 재고 설정 (main.cpp와 동일)
    inventoryRepo.addOrUpdateStock(Inventory("01", 5));  // 콜라 5개
    inventoryRepo.addOrUpdateStock(Inventory("02", 0));  // 사이다 0개
    inventoryRepo.addOrUpdateStock(Inventory("03", 10)); // 녹차 10개
    inventoryRepo.addOrUpdateStock(Inventory("04", 2));  // 홍차 2개

    std::cout << "T1 자판기 재고: 콜라(5), 사이다(0), 녹차(10), 홍차(2)" << std::endl;

    // UC02 완료: 사용자가 콜라(01)를 선택 완료
    std::string selectedDrinkCode = "01";
    std::cout << "UC02 완료 → UC03 시작: 선택된 음료 = " << selectedDrinkCode << " (콜라)" << std::endl;

    // UC03: 현재 자판기 재고 확인 (UserProcessController::state_awaitingDrinkSelection과 동일)
    auto availability = inventoryService.checkDrinkAvailabilityAndPrice(selectedDrinkCode);

    // 재고 확인 결과 검증
    EXPECT_TRUE(availability.isAvailable);    // 구매 가능해야 함
    EXPECT_EQ(availability.currentStock, 5);  // 재고 5개
    EXPECT_GT(availability.price, 0);         // 가격 정보 있어야 함

    std::cout << "재고 확인 결과:" << std::endl;
    std::cout << "  구매 가능: " << (availability.isAvailable ? "YES" : "NO") << std::endl;
    std::cout << "  현재 재고: " << availability.currentStock << "개" << std::endl;
    std::cout << "  가격: " << availability.price << "원" << std::endl;

}
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC03Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from UC03Test
[ RUN      ] UC03Test.CheckStockForAvailableDrink
=== UC03 테스트: 재고 있는 음료 재고 확인 ===
T1 자판기 재고: 콜라(5), 사이다(0), 녹차(10), 홍차(2)
UC02 완료 → UC03 시작: 선택된 음료 = 01 (콜라)
재고 확인 결과:
  구매 가능: YES
  현재 재고: 5개
  가격: 1000원
[       OK ] UC03Test.CheckStockForAvailableDrink (0 ms)
[----------] 1 test from UC03Test (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
```

System Test Case

Use Case 4. 사용자 결제 요청

```
TEST(UC04Test, PaymentSimulationBasicOperation) {
    std::cout << "=== UC04 테스트: 결제 시뮬레이션 기본 동작 ===" << std::endl;

    PaymentCallbackReceiver paymentSim;
    bool paymentResult = false;
    bool callbackCalled = false;

    std::cout << "결제 시뮬레이션 시작 (1초 지연)..." << std::endl;
    auto startTime = std::chrono::steady_clock::now();

    // UC4.3: 결제 시뮬레이션 실행
    paymentSim.simulatePrepayment([&](bool success) {
        paymentResult = success;
        callbackCalled = true;
        std::cout << "결제 시뮬레이션 완료: " << (success ? "성공" : "실패") << std::endl;
    }, 1); // 1초 지연

    auto endTime = std::chrono::steady_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime)

    // 검증
    EXPECT_TRUE(callbackCalled);
    EXPECT_GE(duration.count(), 1000); // 최소 1초 지연 확인
    EXPECT_LT(duration.count(), 1200); // 1.2초 이내 완료 확인

    std::cout << "지연 시간: " << duration.count() << "ms" << std::endl;
    std::cout << "✓ UC04 완료: 결제 시뮬레이션 기본 동작 확인" << std::endl;
}
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC04Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from UC04Test
[ RUN      ] UC04Test.PaymentSimulationBasicOperation
=== UC04 테스트: 결제 시뮬레이션 기본 동작 ===
결제 시뮬레이션 시작 (1초 지연)...
결제 시뮬레이션 완료: 성공
지연 시간: 1003ms
✓ UC04 완료: 결제 시뮬레이션 기본 동작 확인
[       OK ] UC04Test.PaymentSimulationBasicOperation (1003 ms)
[ RUN      ] UC04Test.PaymentConfirmationPrompt
=== UC04 테스트: 결제 확인 프롬프트 ===
결제 프롬프트 표시 테스트...

결제할 금액: 1500원 입니다.
카드를 삽입해주세요. (시뮬레이션: 실제 카드 처리 없음)
결제 승인 중입니다. 잠시만 기다려주세요...
✓ UC04 완료: 결제 프롬프트 메소드 확인
[       OK ] UC04Test.PaymentConfirmationPrompt (0 ms)
[----------] 2 tests from UC04Test (1003 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (1003 ms total)
[  PASSED  ] 2 tests.
```

System Test Case

Use Case 5 결제승인 처리

```
TEST(UC05Test, RegularPurchaseApprovalProcessing) {
    std::cout << "=== UC05 테스트: 일반 구매 결제 승인 처리 ===" << std::endl;

    // Repository 및 Service 초기화
    InventoryRepository inventoryRepo;
    DrinkRepository drinkRepo;
    OrderRepository orderRepo;
    PrepayCodeRepository prepayRepo;
    ErrorService errorService;

    InventoryService inventoryService(inventoryRepo, drinkRepo, errorService);
    PrepaymentService prepaymentService(prepayRepo, orderRepo, errorService);
    OrderService orderService(orderRepo, drinkRepo, inventoryService, prepaymentService, errorService);

    // 초기 재고 설정
    inventoryRepo.addOrUpdateStock(Inventory("01", 5)); // 콜라 5개

    // 주문 생성 (UC2, UC3 완료 후 상태)
    Drink selectedDrink = drinkRepo.findByDrinkCode("01");
    Order testOrder = orderService.createOrder("T1", selectedDrink);

    std::cout << "주문 생성: " << selectedDrink.getName() << " (상태: " << testOrder.getPayStatus() << ")" << std::endl;
    EXPECT_EQ(testOrder.getPayStatus(), "PENDING");

    // 결제 전 재고 확인
    auto beforeStock = inventoryService.checkDrinkAvailabilityAndPrice("01");
    std::cout << "결제 전 재고: " << beforeStock.currentStock << "개" << std::endl;

    // UC5: 결제 승인 처리 (일반 구매)
    orderService.processOrderApproval(testOrder, false); // isPrepayment = false

    // 결과 검증
    EXPECT_EQ(testOrder.getPayStatus(), "APPROVED");

    // 재고 차감 확인
    auto afterStock = inventoryService.checkDrinkAvailabilityAndPrice("01");
    std::cout << "결제 후 재고: " << afterStock.currentStock << "개" << std::endl;
    EXPECT_EQ(afterStock.currentStock, beforeStock.currentStock - 1);

    std::cout << "✓ UC05.1 완료: 일반 구매 결제 승인 처리" << std::endl;
}
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC05Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from UC05Test
[ RUN      ] UC05Test.RegularPurchaseApprovalProcessing
=== UC05 테스트: 일반 구매 결제 승인 처리 ===
주문 생성: 콜라 (상태: PENDING)
결제 전 재고: 5개
결제 후 재고: 4개
✓ UC05.1 완료: 일반 구매 결제 승인 처리
[       OK ] UC05Test.RegularPurchaseApprovalProcessing (0 ms)
[ RUN      ] UC05Test.PrepaymentApprovalProcessing
=== UC05 테스트: 선결제 결제 승인 처리 ===
선결제 주문 생성: 사이다 (상태: PENDING)
생성된 인증코드: hpz2d
✓ UC05.2 완료: 선결제 승인 처리 및 인증코드 생성
[       OK ] UC05Test.PrepaymentApprovalProcessing (0 ms)
[ RUN      ] UC05Test.PaymentApprovalUIDisplay
=== UC05 테스트: 결제 승인 UI 표시 ===
결제 성공 메시지 표시 테스트...
[결제 성공] 결제가 성공적으로 완료되었습니다!
✓ UC05.3 완료: 결제 승인 UI 메시지 표시
[       OK ] UC05Test.PaymentApprovalUIDisplay (0 ms)
[----------] 3 tests from UC05Test (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 3 tests.
```

System Test Case

Use Case 6. 결제거절처리

```
// 테스트 2: 결제 거절 후 UI 메시지 표시
TEST(UC06Test, PaymentDeclinationUIDisplay) {
    std::cout << "=== UC06 테스트: 결제 거절 UI 표시 ===" << std::endl;

    presentation::UserInterface ui;

    // UC6.3: 결제 실패 메시지 표시
    std::cout << "결제 실패 메시지 표시 테스트..." << std::endl;

    EXPECT_NO_THROW({
        ui.displayPaymentResult(false, "결제에 실패했습니다.");
    });

    std::cout << "✓ UC06.2 완료: 결제 거절 UI 메시지 표시" << std::endl;
}
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC06Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from UC06Test
[ RUN      ] UC06Test.PaymentDeclinationProcessing
=== UC06 테스트: 결제 거절 처리 ===
주문 생성: 콜라 (상태: PENDING)
결제 전 재고: 5개
결제 후 재고: 5개
✓ UC06.1 완료: 결제 거절 처리
[       OK ] UC06Test.PaymentDeclinationProcessing (0 ms)
[ RUN      ] UC06Test.PaymentDeclinationUIDisplay
=== UC06 테스트: 결제 거절 UI 표시 ===
결제 실패 메시지 표시 테스트...
[결제 실패] 결제에 실패했습니다.
✓ UC06.2 완료: 결제 거절 UI 메시지 표시
[       OK ] UC06Test.PaymentDeclinationUIDisplay (0 ms)
[----------] 2 tests from UC06Test (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

System Test Case

Use Case 7. 음료배출제어

```
TEST(UC07Test, StockReductionLogic) {
    std::cout << "=== UC07 테스트: 재고 차감 로직 상세 ===" << std::endl;

    // Repository 및 Service 초기화
    InventoryRepository inventoryRepo;
    DrinkRepository drinkRepo;
    ErrorService errorService;
    InventoryService inventoryService(inventoryRepo, drinkRepo, errorService);

    // 초기 재고 설정
    inventoryRepo.addOrUpdateStock(Inventory("01", 3)); // 콜라 3개

    std::cout << "초기 재고: 3개" << std::endl;
    auto initialStock = inventoryService.checkDrinkAvailabilityAndPrice("01");
    EXPECT_EQ(initialStock.currentStock, 3);
    EXPECT_TRUE(initialStock.isAvailable);

    // UC7.2: 일반 구매 재고 차감 (OrderService::processOrderApproval에서 수행)
    inventoryService.decreaseStock("01");

    auto afterFirstDecrease = inventoryService.checkDrinkAvailabilityAndPrice("01");
    std::cout << "1차 차감 후 재고: " << afterFirstDecrease.currentStock << "개" << std::endl;
    EXPECT_EQ(afterFirstDecrease.currentStock, 2);
    EXPECT_TRUE(afterFirstDecrease.isAvailable);

    // 추가 차감
    inventoryService.decreaseStock("01");
    inventoryService.decreaseStock("01");

    auto afterAllDecrease = inventoryService.checkDrinkAvailabilityAndPrice("01");
    std::cout << "전체 차감 후 재고: " << afterAllDecrease.currentStock << "개" << std::endl;
    EXPECT_EQ(afterAllDecrease.currentStock, 0);
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC07Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from UC07Test
[ RUN      ] UC07Test.DrinkDispensingUIMessages
=== UC07 테스트: 음료 배출 UI 메시지 ===
배출 시작 메시지 테스트...

[콜라] 음료가 배출되고 있습니다. 잠시만 기다려주세요...
배출 완료 메시지 테스트...
[콜라] 음료가 나왔습니다. 이용해주셔서 감사합니다!
✓ UC07.3 완료: 음료 배출 UI 메시지 테스트
[       OK ] UC07Test.DrinkDispensingUIMessages (0 ms)
[ RUN      ] UC07Test.StockReductionLogic
=== UC07 테스트: 재고 차감 로직 상세 ===
초기 재고: 3개
1차 차감 후 재고: 2개
전체 차감 후 재고: 0개
✓ UC07.4 완료: 재고 차감 로직 검증
[       OK ] UC07Test.StockReductionLogic (0 ms)
[----------] 2 tests from UC07Test (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

System Test Case

Use Case 8. 재고조회브로드캐스트

```
TEST(UC08_UC09Test, MultipleVendingMachinesNetwork) {

    // T1 자판기 (요청자)
    std::vector<std::string> t1_endpoints = {"127.0.0.1:12346", "127.0.0.1:12347"};
    std::unordered_map<std::string, std::string> t1_idMap = {
        {"T2", "127.0.0.1:12346"}, {"T3", "127.0.0.1:12347"}
    };
    VendingMachineSimulator t1("T1", 10, 10, 12345, t1_endpoints, t1_idMap);
    t1.setupInventory("02", 0); // 사이다 재고 없음

    // T2 자판기 (응답자 1 - 재고 있음)
    std::vector<std::string> t2_endpoints = {"127.0.0.1:12345"};
    std::unordered_map<std::string, std::string> t2_idMap = {{"T1", "127.0.0.1:12345"}};
    VendingMachineSimulator t2("T2", 20, 20, 12346, t2_endpoints, t2_idMap);
    t2.setupInventory("02", 3); // 사이다 3개

    // T3 자판기 (응답자 2 - 재고 있음)
    std::vector<std::string> t3_endpoints = {"127.0.0.1:12345"};
    std::unordered_map<std::string, std::string> t3_idMap = {{"T1", "127.0.0.1:12345"}};
    VendingMachineSimulator t3("T3", 30, 30, 12347, t3_endpoints, t3_idMap);
    t3.setupInventory("02", 7); // 사이다 7개

    std::cout << "\n다중 자판기 시나리오:" << std::endl;
    std::cout << "- T1(12345): 사이다 재고 0개 → 브로드캐스트 요청" << std::endl;
    std::cout << "- T2(12346): 사이다 재고 3개 → 응답 예정" << std::endl;
    std::cout << "- T3(12347): 사이다 재고 7개 → 응답 예정" << std::endl;

    // 모든 자판기 네트워킹 시작 (응답자들 먼저)
    t2.startNetworking();
    t3.startNetworking();
    std::this_thread::sleep_for(std::chrono::milliseconds(400));
    t1.startNetworking();
    std::this_thread::sleep_for(std::chrono::milliseconds(400));

    // UC8: T1에서 사이다 재고 조회 브로드캐스트
    std::cout << "\nUC08 실행: T1 → 다중 브로드캐스트 재고 조회 (사이다)" << std::endl;
    t1.sendStockRequest("02");
```

```
UC08 실행 : T1 → 다중 브로드캐스트 재고 조회 (사이다)
[T1] 재고 조회 브로드캐스트 전송 : 02
[Sender DEBUG] Sending to 127.0.0.1:12346: {"msg_type":0,"src_id":"T1","dst_id":"0","msg_content":{"item_num":"1","item_code":"02"}}

 Accepting connections on port 12346
[Receiver DEBUG] Received line: {"msg_type":0,"src_id":"T1","dst_id":"0","msg_content":{"item_num":"1","item_code":"02"}}
[Sender DEBUG] Sending to 127.0.0.1:12347: {"msg_type":0,"src_id":"T1","dst_id":"0","msg_content":{"item_num":"1","item_code":"02"}}

UC09 대기 : 다중 자판기 응답 처리...
[Receiver DEBUG] Parsed msg type: 0 from T1
[T2] REQ_STOCK 수신 : T1
 Accepting connections on port 12347
[Receiver DEBUG] Received line: {"msg_type":0,"src_id":"T1","dst_id":"0","msg_content":{"item_num":"1","item_code":"02"}}
[Receiver DEBUG] Parsed msg type: 0 from T1
[T3] REQ_STOCK 수신 : T1
[Sender DEBUG] Sending to 127.0.0.1:12345: {"msg_type":1,"src_id":"T2","dst_id":"T1","msg_content":{"coor_y":"20","item_num":"3","coor_x":"20","item_
"02"}}

[T2] RESP_STOCK 전송 : T1 (재고 : 3)
 Accepting connections on port 12345
[Receiver DEBUG] Received line: {"msg_type":1,"src_id":"T2","dst_id":"T1","msg_content":{"coor_y":"20","item_num":"3","coor_x":"20","item_code":"02"}}
[Receiver DEBUG] Parsed msg type: 1 from T2
[T1] RESP_STOCK 수신 : T2
[Sender DEBUG] Sending to 127.0.0.1:12345: {"msg_type":1,"src_id":"T3","dst_id":"T1","msg_content":{"coor_y":"30","item_num":"7","coor_x":"30","item_
"02"}}

[T3] RESP_STOCK 전송 : T1 (재고 : 7)
 Accepting connections on port 12345
[Receiver DEBUG] Received line: {"msg_type":1,"src_id":"T3","dst_id":"T1","msg_content":{"coor_y":"30","item_num":"7","coor_x":"30","item_code":"02"}}
[Receiver DEBUG] Parsed msg type: 1 from T3
[T1] RESP_STOCK 수신 : T3
✓ UC08 완료 : T1에서 다중 브로드캐스트 전송
✓ UC09 완료 : T2 응답 (재고 : 3개 )
✓ UC09 완료 : T3 응답 (재고 : 7개 )
[T1] 종료 시작...
[T1] io_context 스레드 종료
[T1] 종료 완료
[T2] 종료 시작...
[T2] io_context 스레드 종료
[T2] 종료 완료
[T3] 종료 시작...
[T3] io_context 스레드 종료
[T3] 종료 완료
[       OK ] UC08_UC09Test.MultipleVendingMachinesNetwork (4130 ms)
[ RUN      ] UC08_UC09Test.NoStockResponseTest
=== UC08+UC09 테스트 : 재고 없는 경우 응답 ===
```

System Test Case

Use Case 9. 재고 조회 브로드캐스트에대한 다른 자판기들의 응답 수신

```
TEST(UC08_UC09Test, NoStockResponseTest) {
    std::cout << "=== UC08+UC09 테스트: 재고 없는 경우 응답 ===" << std::endl;

    // T1 자판기 (요청자)
    std::vector<std::string> t1_endpoints = {"127.0.0.1:12346"};
    std::unordered_map<std::string, std::string> t1_idMap = {{"T2", "127.0.0.1:12346"}};
    VendingMachineSimulator t1("T1", 10, 10, 12345, t1_endpoints, t1_idMap);
    t1.setupInventory("03", 0); // 녹차 재고 없음

    // T2 자판기 (응답자 – 역시 재고 없음)
    std::vector<std::string> t2_endpoints = {"127.0.0.1:12345"};
    std::unordered_map<std::string, std::string> t2_idMap = {{"T1", "127.0.0.1:12345"}};
    VendingMachineSimulator t2("T2", 20, 20, 12346, t2_endpoints, t2_idMap);
    t2.setupInventory("03", 0); // 녹차 재고 없음

    std::cout << "\n재고 없음 시나리오:" << std::endl;
    std::cout << "– T1: 녹차 재고 0개" << std::endl;
    std::cout << "– T2: 녹차 재고 0개 → 재고 없음 응답 예정" << std::endl;

    // 네트워킹 시작
    t2.startNetworking();
    std::this_thread::sleep_for(std::chrono::milliseconds(300));
    t1.startNetworking();
    std::this_thread::sleep_for(std::chrono::milliseconds(300));

    // UC8+UC9: 재고 조회 및 재고 없음 응답
    std::cout << "\nUC08+UC09 실행: 재고 없음 시나리오 테스트" << std::endl;
    t1.sendStockRequest("03");

    // 응답 처리 대기
    std::this_thread::sleep_for(std::chrono::seconds(2));

    // 두 자판기 모두 재고 없음 확인
    auto t1_availability = t1.getAvailability("03");
    auto t2_availability = t2.getAvailability("03");

    EXPECT_FALSE(t1_availability.isAvailable);
    EXPECT_EQ(t1_availability.currentStock, 0);
    EXPECT_FALSE(t2_availability.isAvailable);
```

```
    // UC9: 다중 응답 처리 대기
    std::cout << "UC09 대기: 다중 자판기 응답 처리..." << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(3));

    // 각 응답자 자판기의 재고 확인
    auto t2_availability = t2.getAvailability("02");
    auto t3_availability = t3.getAvailability("02");

    EXPECT_TRUE(t2_availability.isAvailable);
    EXPECT_EQ(t2_availability.currentStock, 3);
    EXPECT_TRUE(t3_availability.isAvailable);
    EXPECT_EQ(t3_availability.currentStock, 7);

    std::cout << "✓ UC08 완료: T1에서 다중 브로드캐스트 전송" << std::endl;
    std::cout << "✓ UC09 완료: T2 응답 (재고: " << t2_availability.currentStock << "개)" << std::endl;
    std::cout << "✓ UC09 완료: T3 응답 (재고: " << t3_availability.currentStock << "개)" << std::endl;

    // 정리
    t1.shutdown();
    t2.shutdown();
    t3.shutdown();
}
```

```
재고 없음 시나리오:
– T1: 녹차 재고 0개
– T2: 녹차 재고 0개 → 재고 없음 응답 예정
[T2] 네트워킹 시작...
 Accepting connections on port 12346
[T2] io_context 스레드 시작
[T2] 네트워킹 시작 완료
[T1] 네트워킹 시작...
 Accepting connections on port 12345
[T1] io_context 스레드 시작
[T1] 네트워킹 시작 완료

UC08+UC09 실행: 재고 없음 시나리오 테스트
[T1] 재고 조회 브로드캐스트 전송: 03
[Sender DEBUG] Sending to 127.0.0.1:12346: {"msg_type":0,"src_id":"T1","dst_id":"0","msg_content":{"item_num":"1","item_code":"03"}}

 Accepting connections on port 12346
[Receiver DEBUG] Received line: {"msg_type":0,"src_id":"T1","dst_id":"0","msg_content":{"item_num":"1","item_code":"03"}}
[Receiver DEBUG] Parsed msg type: 0 from T1
[T2] REQ_STOCK 수신: T1
[Sender DEBUG] Sending to 127.0.0.1:12345: {"msg_type":1,"src_id":"T2","dst_id":"T1","msg_content":{"coor_y":"20","item_num":"0","coor_x":"20","item_code":
"03"}}

 Accepting connections on port 12345
[T2] RESP_STOCK 전송: T1 (재고: 0)
[Receiver DEBUG] Received line: {"msg_type":1,"src_id":"T2","dst_id":"T1","msg_content":{"coor_y":"20","item_num":"0","coor_x":"20","item_code":"03"}}
[Receiver DEBUG] Parsed msg type: 1 from T2
[T1] RESP_STOCK 수신: T2
✓ UC08+UC09 완료: 재고 없음 케이스 처리
[T1] 종료 시작...
[T1] io_context 스레드 종료
[T1] 종료 완료
[T2] 종료 시작...
[T2] io_context 스레드 종료
[T2] 종료 완료
[       OK ] UC08_UC09Test.NoStockResponseTest (2819 ms)
[----------] 2 tests from UC08_UC09Test (6950 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (6950 ms total)
[  PASSED  ] 2 tests.
baikhyunjun@baeghyeonjun-ui-MacBookPro build %
```

Ln 240, Col 5 Spaces: 4 UTF-8 LF {} C++ Win32

System Test Case

Use Case 10. 가장 가까운 자판기 안내

```
TEST(SystemTest, BroadcastAndFindNearestVendingMachine) {
    // 1. 현재 자판기(t1) 정보
    int t1_x = 0, t1_y = 0;
    VendingMachine t1("T1", t1_x, t1_y, "12345");

    // 2. 브로드캐스트로 응답받은 다른 자판기 정보(t2, t3)
    std::vector<OtherVendingMachineInfo> responses;
    responses.push_back({"T2", 10, 10, true}); // t2: (10,0)
    responses.push_back({"T3", 5, 5, true});  // t3: (5,5)

    // 3. 거리 계산 및 가장 가까운 자판기 찾기
    DistanceService distanceService;
    auto nearestOpt = distanceService.findNearestAvailableVendingMachine(t1_x, t1_y, responses);

    ASSERT_TRUE(nearestOpt.has_value());
    VendingMachine nearest = nearestOpt.value();

    // 4. 사용자 안내 메시지(예상 결과)
    EXPECT_EQ(nearest.getId(), "T3");
    EXPECT_EQ(nearest.getLocation(), std::make_pair(5, 5));
}
```

```
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from SystemTest
[ RUN      ] SystemTest.BroadcastAndFindNearestVendingMachine
[       OK ] SystemTest.BroadcastAndFindNearestVendingMachine (0 ms)
[----------] 1 test from SystemTest (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
```

System Test Case

Use Case 11. 선결제 요청

```cpp
TEST(UC11Test, DisplayMessageFunctionality) {
    std::cout << "=== UC11 테스트: displayMessage 기능 테스트 ===" << std::endl;

    // 테스트할 메시지들
    std::vector<std::string> messages = {
        "선결제를 진행하시겠습니까?",
        "결제가 완료되었습니다.",
        "인증코드: ABC12345",
        "선결제가 취소되었습니다.",
        "오류가 발생했습니다."
    };

    for (size_t i = 0; i < messages.size(); i++) {
        // 실제 테스트: 메시지 유효성 검증
        EXPECT_FALSE(messages[i].empty());
        EXPECT_GT(messages[i].length(), 0);

        // 한글이 포함된 메시지인지 확인 (선결제 관련 메시지이므로)
        bool hasKorean = false;
        for (unsigned char c : messages[i]) {
            if (c >= 0xAC && c <= 0xD7) {
                hasKorean = true;
                break;
            }
        }
        EXPECT_TRUE(hasKorean || messages[i].find("ABC") != std::string::npos); // 한글이거나 인증코드
    }

    std::cout << "\n✓ displayMessage 기능 테스트 완료" << std::endl;
}
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC11Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from UC11Test
[ RUN      ] UC11Test.DisplayMessageFunctionality
=== UC11 테스트: displayMessage 기능 테스트 ===

✓ displayMessage 기능 테스트 완료
[       OK ] UC11Test.DisplayMessageFunctionality (0 ms)
[----------] 1 test from UC11Test (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.
```

System Test Case

Use Case 12. 인증코드 발급 및 위치 안내

```
TEST(UC12Test, AuthCodeGeneration) {
    persistence::PrepayCodeRepository prepayRepo;
    persistence::OrderRepository orderRepo;
    service::ErrorService errorService;
    service::PrepaymentService prepaymentService(prepayRepo, orderRepo, errorService);

    // 인증코드 생성
    std::string authCode = prepaymentService.generateAuthCodeString();

    // 인증코드가 5자리 영숫자인지 확인
    EXPECT_EQ(authCode.length(), 5);
```

```
// 테스트 2a: 가까운 자판기 찾기 테스트 - 사용자 위치 (20,20)
TEST(UC12Test, FindNearestVendingMachine_Location20_20) {
    // 사용자 위치 설정 (20, 20)
    int userX = 20, userY = 20;

    // 가장 가까운 자판기 찾기
    VendingMachine nearest = findNearestVendingMachine(userX, userY, TEST_VENDING_MACHINES);

    // 가장 가까운 자판기가 찾아졌는지 확인
    EXPECT_FALSE(nearest.getId().empty());

    // 거리 계산
```

```
// 테스트 2b: 가까운 자판기 찾기 테스트 - 사용자 위치 (30,30)
TEST(UC12Test, FindNearestVendingMachine_Location30_30) {
    // 사용자 위치 설정 (30, 30)
    int userX = 30, userY = 30;

    // 가장 가까운 자판기 찾기
    VendingMachine nearest = findNearestVendingMachine(userX, userY, TEST_VENDING_MACHINES);

    // 가장 가까운 자판기가 찾아졌는지 확인
    EXPECT_FALSE(nearest.getId().empty());

    // 거리 계산
    double distance = calculateDistance(userX, userY, nearest.getLocation().first, nearest.getLocation().second);

    // T2(20,20)가 가장 가까움 (거리 약 14.14)
    EXPECT_NEAR(distance, 14.14, 0.1);
    EXPECT_EQ(nearest.getId(), "T2");
}
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC12Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googlete
[==========] Running 6 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 6 tests from UC12Test
[ RUN      ] UC12Test.AuthCodeGeneration
[       OK ] UC12Test.AuthCodeGeneration (0 ms)
[ RUN      ] UC12Test.FindNearestVendingMachine_Location20_20
[       OK ] UC12Test.FindNearestVendingMachine_Location20_20 (0 ms)
[ RUN      ] UC12Test.FindNearestVendingMachine_Location30_30
[       OK ] UC12Test.FindNearestVendingMachine_Location30_30 (0 ms)
[ RUN      ] UC12Test.VendingMachineInventoryCheck
[       OK ] UC12Test.VendingMachineInventoryCheck (0 ms)
[ RUN      ] UC12Test.PrepaymentRegistration
[       OK ] UC12Test.PrepaymentRegistration (0 ms)
[ RUN      ] UC12Test.IntegratedScenario
✓ 인증코드 9EpPe 발급 완료
✓ 가장 가까운 자판기: T2 (위치: 20, 20)
[       OK ] UC12Test.IntegratedScenario (0 ms)
[----------] 6 tests from UC12Test (0 ms total)
```

System Test Case

Use Case 13. 인증코드 입력 처리

```
TEST(UC13Test, AuthCodeInputValidation_InvalidCases) {
    persistence::PrepayCodeRepository prepayRepo;
    persistence::OrderRepository orderRepo;
    service::ErrorService errorService;
    service::PrepaymentService prepaymentService(prepayRepo, orderRepo, errorService);

    // 1. 잘못된 형식 - 길이가 짧음 (4자리)
    std::string shortCode = "AB12";
    EXPECT_FALSE(prepaymentService.isValidAuthCodeFormat(shortCode));

    // 2. 잘못된 형식 - 길이가 김 (6자리)
    std::string longCode = "AB1234";
    EXPECT_FALSE(prepaymentService.isValidAuthCodeFormat(longCode));

    // 3. 잘못된 형식 - 특수문자 포함
    std::string specialCharCode = "AB1@#";
    EXPECT_FALSE(prepaymentService.isValidAuthCodeFormat(specialCharCode));

    // 4. 잘못된 형식 - 공백 포함
    std::string spaceCode = "AB 12";
    EXPECT_FALSE(prepaymentService.isValidAuthCodeFormat(spaceCode));

    // 5. 잘못된 형식 - 빈 문자열
    std::string emptyCode = "";
    EXPECT_FALSE(prepaymentService.isValidAuthCodeFormat(emptyCode));

    // 6. 형식은 맞지만 존재하지 않는 인증코드
    std::string nonExistentCode = "XYZ99";
    EXPECT_TRUE(prepaymentService.isValidAuthCodeFormat(nonExistentCode)); // 형식은 올바름

    PrePaymentCode result = prepaymentService.getPrepaymentDetailsIfActive(nonExistentCode);
    EXPECT_TRUE(result.getCode().empty()); // 존재하지 않으므로 빈 객체 반환
}
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC13Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 5 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 5 tests from UC13Test
[ RUN      ] UC13Test.AuthCodeInputValidation_InvalidCases
[       OK ] UC13Test.AuthCodeInputValidation_InvalidCases (0 ms)
[ RUN      ] UC13Test.AuthCodeInputValidation_ValidCase
[       OK ] UC13Test.AuthCodeInputValidation_ValidCase (0 ms)
[ RUN      ] UC13Test.AuthCodeInputValidation_UsedCode
[       OK ] UC13Test.AuthCodeInputValidation_UsedCode (0 ms)
[ RUN      ] UC13Test.AuthCodeFormat_VariousValidFormats
[       OK ] UC13Test.AuthCodeFormat_VariousValidFormats (0 ms)
[ RUN      ] UC13Test.IntegratedAuthCodeInputProcess
발급된 인증코드: kfe17
잘못된 입력 '123': 형식 검증 실패
잘못된 입력 'ABC@#': 형식 검증 실패
올바른 코드 'kfe17': 검증 성공, 음료 제공 가능
코드 'kfe17': 사용 완료, 더 이상 유효하지 않음
✓ 인증코드 입력 검증 프로세스 테스트 완료
[       OK ] UC13Test.IntegratedAuthCodeInputProcess (0 ms)
[----------] 5 tests from UC13Test (0 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 5 tests.
baikhyunjun@baeghyeonjun-ui-MacBookPro build %
```

System Test Case

Use Case 14. 인증코드 유효성 검증

```
TEST(UC14Test, ProperUsageAndStatusChangeToUsed) {
    persistence::PrepayCodeRepository prepayRepo;
    persistence::OrderRepository orderRepo;
    service::ErrorService errorService;
    service::PrepaymentService prepaymentService(prepayRepo, orderRepo, errorService);

    // 1. 유효한 인증코드 생성 및 등록
    auto order = std::make_shared<Order>("T2", "02", 1, "", "APPROVED");
    std::string authCode = prepaymentService.generateAuthCodeString();
    order->setCertCode(authCode);
    prepaymentService.registerPrepayment(authCode, order);

    std::cout << "인증코드 생성: " << authCode << " (음료코드: " << order->getDrinkCode() << ")" << std::endl;

    // 2. 사용 전 상태 확인 – 활성 상태여야 함
    PrePaymentCode beforeUse = prepaymentService.getPrepaymentDetailsIfActive(authCode);
    EXPECT_FALSE(beforeUse.getCode().empty());
    EXPECT_TRUE(beforeUse.isUsable());

    auto linkedOrder = beforeUse.getHeldOrder();
    ASSERT_NE(linkedOrder, nullptr);
    EXPECT_EQ(linkedOrder->getDrinkCode(), "02");

    std::cout << "사용 전 상태: 코드 활성화됨, 주문 정보 조회 가능" << std::endl;

    // 3. 정상적인 음료 제공 과정 시뮬레이션
    std::cout << "음료 제공 중..." << std::endl;

    // 4. 음료 제공 완료 후 인증코드를 사용됨 상태로 변경
    prepaymentService.changeAuthCodeStatusToUsed(authCode);
    std::cout << "음료 제공 완료, 인증코드 상태를 'used'로 변경" << std::endl;
```

```
    auto linkedOrder = beforeUse.getHeldOrder();
    ASSERT_NE(linkedOrder, nullptr);
    EXPECT_EQ(linkedOrder->getDrinkCode(), "02");

    std::cout << "사용 전 상태: 코드 활성화됨, 주문 정보 조회 가능" << std::endl;

    // 3. 정상적인 음료 제공 과정 시뮬레이션
    std::cout << "음료 제공 중..." << std::endl;

    // 4. 음료 제공 완료 후 인증코드를 사용됨 상태로 변경
    prepaymentService.changeAuthCodeStatusToUsed(authCode);
    std::cout << "음료 제공 완료, 인증코드 상태를 'used'로 변경" << std::endl;

    // 5. 사용 후 상태 확인 – Repository에서 직접 상태를 확인
    PrePaymentCode afterUse = prepayRepo.findByCode(authCode);
    EXPECT_FALSE(afterUse.getCode().empty()); // 코드는 여전히 존재함
    EXPECT_EQ(afterUse.getStatus(), domain::CodeStatus::USED); // 상태가 USED로 변경되었는지 확인
    EXPECT_FALSE(afterUse.isUsable()); // 더 이상 사용 불가능해야 함

    std::cout << "사용 후 상태: 코드 상태가 'USED'로 정확히 변경됨" << std::endl;
    std::cout << "✓ 테스트 3 완료: 정상 사용 후 상태가 USED로 변경되었음을 직접 확인" << std::endl;
}
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC14Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 3 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 3 tests from UC14Test
[ RUN      ] UC14Test.ValidFormatButNonExistentAuthCode
형식 검증 통과 : ABCD1
존재하지 않는 코드 'ABCD1': 검증 실패, 코드를 찾을 수 없음
✓ 테스트 1 완료 : 형식은 유효하나 존재하지 않는 코드 처리
[       OK ] UC14Test.ValidFormatButNonExistentAuthCode (0 ms)
[ RUN      ] UC14Test.AlreadyUsedAuthCodeVerification
인증코드 생성 및 등록 : S0uli
첫 번째 사용 완료, 코드 상태를 'used'로 변경
두 번째 사용 시도 : 코드 'S0uli'는 이미 사용되어 무효함
✓ 테스트 2 완료 : 이미 사용된 코드 재사용 방지
[       OK ] UC14Test.AlreadyUsedAuthCodeVerification (0 ms)
[ RUN      ] UC14Test.ProperUsageAndStatusChangeToUsed
인증코드 생성 : qtgFs (음료코드 : 02)
사용 전 상태 : 코드 활성화됨, 주문 정보 조회 가능
음료 제공 중...
음료 제공 완료, 인증코드 상태를 'used'로 변경
사용 후 상태 : 코드 상태가 'USED'로 정확히 변경됨
✓ 테스트 3 완료 : 정상 사용 후 상태가 USED로 변경되었음을 직접 확인
[       OK ] UC14Test.ProperUsageAndStatusChangeToUsed (0 ms)
[----------] 3 tests from UC14Test (0 ms total)

[----------] Global test environment tear-down
[==========] 3 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 3 tests.
```

System Test Case

Use Case 15. 선결제 요청 수신 및 재고 확보

```
TEST(UC15Test, AcceptPrepaymentRequestWithSufficientStock) {
    boost::asio::io_context io_context;
    std::vector<std::string> empty_endpoints;
    std::unordered_map<std::string, std::string> empty_id_map;

    // Repository 및 Service 초기화
    persistence::InventoryRepository inventoryRepo;
    persistence::DrinkRepository drinkRepo;
    persistence::PrepayCodeRepository prepayRepo;
    persistence::OrderRepository orderRepo;
    service::ErrorService errorService;
    service::InventoryService inventoryService(inventoryRepo, drinkRepo, errorService);
    service::PrepaymentService prepaymentService(prepayRepo, orderRepo, errorService);

    network::MessageSender messageSender(io_context, empty_endpoints, empty_id_map);
    network::MessageReceiver messageReceiver(io_context, 12345);
    service::MessageService messageService(messageSender, messageReceiver, errorService, "T2", 20, 20);

    // 초기 재고 설정 (T2 자판기: 사이다 12개)
    inventoryRepo.addOrUpdateStock(Inventory("02", 12)); // 사이다 12개

    // 사용 전 재고 확인
    auto beforeStock = inventoryService.checkDrinkAvailabilityAndPrice("02");
    EXPECT_TRUE(beforeStock.isAvailable);
    EXPECT_EQ(beforeStock.currentStock, 12);

    std::cout << "T2 자판기 초기 상태: 사이다 " << beforeStock.currentStock << "개" << std::endl;
```

```
    // T1에서 보낸 선결제 요청 메시지 시뮬레이션
    network::Message prepayRequest;
    prepayRequest.msg_type = network::Message::Type::REQ_PREPAY;
    prepayRequest.src_id = "T1";
    prepayRequest.dst_id = "T2";
    prepayRequest.msg_content["item_code"] = "02"; // 사이다
    prepayRequest.msg_content["item_num"] = "1";
    prepayRequest.msg_content["cert_code"] = "ABC12";

    std::cout << "T1에서 T2로 선결제 요청: 사이다(02), 인증코드 ABC12" << std::endl;

    // 재고 확인 및 차감
    auto availability = inventoryService.checkDrinkAvailabilityAndPrice("02");
    bool reservationSuccess = false;

    if (availability.isAvailable && availability.currentStock >= 1) {
        inventoryService.decreaseStock("02"); // 재고 1개 차감
        prepaymentService.recordIncomingPrepayment("ABC12", "02", "T1");
        reservationSuccess = true;
        std::cout << "재고 확보 성공: 사이다 1개 예약, 재고 차감" << std::endl;
    }

    EXPECT_TRUE(reservationSuccess);

    // 재고 차감 확인
    auto afterStock = inventoryService.checkDrinkAvailabilityAndPrice("02");
    EXPECT_EQ(afterStock.currentStock, 11); // 12 - 1 = 11

    // 선결제 정보 저장 확인
    PrePaymentCode savedPrepay = prepayRepo.findByCode("ABC12");
    EXPECT_FALSE(savedPrepay.getCode().empty());
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC15Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from UC15Test
[ RUN      ] UC15Test.AcceptPrepaymentRequestWithSufficientStock
T2 자판기 초기 상태: 사이다 12개
T1에서 T2로 선결제 요청: 사이다(02), 인증코드 ABC12
재고 확보 성공: 사이다 1개 예약, 재고 차감
✓ 테스트 1 완료: 충분한 재고로 선결제 요청 수락, 재고 차감 및 선결제 정보 저장
[       OK ] UC15Test.AcceptPrepaymentRequestWithSufficientStock (0 ms)
[ RUN      ] UC15Test.RejectPrepaymentRequestWithInsufficientStock
T1 자판기 초기 상태: 사이다 재고 없음 (0개)
T3에서 T1로 선결제 요청: 사이다(02), 인증코드 XYZ99
재고 부족으로 선결제 요청 거절
✓ 테스트 2 완료: 재고 부족으로 선결제 요청 거절, 재고 및 데이터 변경 없음
[       OK ] UC15Test.RejectPrepaymentRequestWithInsufficientStock (0 ms)
[----------] 2 tests from UC15Test (0 ms total)
```

System Test Case

Use Case 16. 재고 확보 요청 전송

```cpp
TEST(UC16Test, CreatePrepaymentReservationRequest) {
    std::cout << "=== UC16 테스트: 재고 확보 요청 메시지 생성 ===" << std::endl;

    // 선결제 주문 정보 (이미 생성되어 있다고 가정)
    std::string authCode = "ABC12";
    std::string drinkCode = "02"; // 사이다
    std::string targetVmId = "T2"; // 목표 자판기

    std::cout << "선결제 정보: 인증코드=" << authCode << ", 음료=" << drinkCode << ", 목표VM=" << targetVmId << std::endl;

    // 재고 확보 요청 메시지 생성
    network::Message reservationRequest;
    reservationRequest.msg_type = network::Message::Type::REQ_PREPAY;
    reservationRequest.src_id = "T1";
    reservationRequest.dst_id = targetVmId;
    reservationRequest.msg_content["item_code"] = drinkCode;
    reservationRequest.msg_content["item_num"] = "1";
    reservationRequest.msg_content["cert_code"] = authCode;

    // 메시지 내용 검증
    EXPECT_EQ(reservationRequest.msg_type, network::Message::Type::REQ_PREPAY);
    EXPECT_EQ(reservationRequest.src_id, "T1");
    EXPECT_EQ(reservationRequest.dst_id, "T2");
    EXPECT_EQ(reservationRequest.msg_content.at("item_code"), "02");
    EXPECT_EQ(reservationRequest.msg_content.at("item_num"), "1");
    EXPECT_EQ(reservationRequest.msg_content.at("cert_code"), "ABC12");

    std::cout << "재고 확보 요청 메시지: " << reservationRequest.src_id
              << " -> " << reservationRequest.dst_id
              << " (음료: " << reservationRequest.msg_content.at("item_code") << ")" << std::endl;
    std::cout << "✓ 테스트 1 완료: REQ_PREPAY 메시지 생성 성공" << std::endl;
}
```

```cpp
TEST(UC16Test, SendMultiplePrepaymentRequests) {
    std::cout << "=== UC16 테스트: 여러 자판기에 재고 확보 요청 ===" << std::endl;

    // 공통 선결제 정보
    std::string authCode = "MULTI";
    std::string drinkCode = "01"; // 콜라

    // 여러 목표 자판기 목록
    std::vector<std::string> targetVms = {"T2", "T3", "T5"};

    std::cout << "선결제 정보: 인증코드=" << authCode << ", 음료=" << drinkCode << std::endl;
    std::cout << "목표 자판기 목록: ";
    for (const auto& vm : targetVms) {
        std::cout << vm << " ";
    }
    std::cout << std::endl;

    // 각 자판기에 요청 메시지 생성
    std::vector<network::Message> requests;
    for (const std::string& targetVm : targetVms) {
        network::Message request;
        request.msg_type = network::Message::Type::REQ_PREPAY;
        request.src_id = "T1";
        request.dst_id = targetVm;
        request.msg_content["item_code"] = drinkCode;
        request.msg_content["item_num"] = "1";
        request.msg_content["cert_code"] = authCode;

        requests.push_back(request);

        // 개별 메시지 검증
        EXPECT_EQ(request.dst_id, targetVm);
        EXPECT_EQ(request.msg_content.at("item_code"), drinkCode);
```

```
baikhyunjun@baeghyeonjun-ui-MacBookPro build % ./runUC16Test.out
Running main() from /Users/baikhyunjun/gitclone/SMA_T6/lib/googletest/googletest/src/gtest_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from UC16Test
[ RUN      ] UC16Test.CreatePrepaymentReservationRequest
=== UC16 테스트: 재고 확보 요청 메시지 생성 ===
선결제 정보: 인증코드=ABC12, 음료=02, 목표VM=T2
재고 확보 요청 메시지: T1 -> T2 (음료: 02)
✓ 테스트 1 완료: REQ_PREPAY 메시지 생성 성공
[       OK ] UC16Test.CreatePrepaymentReservationRequest (0 ms)
[ RUN      ] UC16Test.SendMultiplePrepaymentRequests
=== UC16 테스트: 여러 자판기에 재고 확보 요청 ===
선결제 정보: 인증코드=MULTI, 음료=01
목표 자판기 목록: T2 T3 T5
요청 생성: T1 -> T2 (콜라 1개)
요청 생성: T1 -> T3 (콜라 1개)
요청 생성: T1 -> T5 (콜라 1개)
✓ 테스트 3 완료: 여러 자판기에 대한 재고 확보 요청 생성
[       OK ] UC16Test.SendMultiplePrepaymentRequests (0 ms)
[----------] 2 tests from UC16Test (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

System Test Case

Use Case 17. 타 자판기로부터 재고 조회 요청 수신 및 응답 처리

```
TEST(UC17Test, ReceiveStockInquiryAndCheckInventory) {
    std::cout << "=== UC17 테스트: 재고 조회 요청 수신 및 재고 확인 ===" << std::endl;

    // Repository 및 Service 초기화
    persistence::InventoryRepository inventoryRepo;
    persistence::DrinkRepository drinkRepo;
    service::ErrorService errorService;
    service::InventoryService inventoryService(inventoryRepo, drinkRepo, errorService);

    // T2 자판기 재고 설정 (main.cpp 참조)
    inventoryRepo.addOrUpdateStock(Inventory("01", 0));  // 콜라 재고 없음
    inventoryRepo.addOrUpdateStock(Inventory("02", 12)); // 사이다 12개
    inventoryRepo.addOrUpdateStock(Inventory("08", 8));  // 캔커피 8개
    inventoryRepo.addOrUpdateStock(Inventory("09", 15)); // 물 15개

    std::cout << "T2 자판기 재고: 콜라(0), 사이다(12), 캔커피(8), 물(15)" << std::endl;

    // T1에서 보낸 재고 조회 요청 메시지 시뮬레이션
    network::Message stockRequest;
    stockRequest.msg_type = network::Message::Type::REQ_STOCK;
    stockRequest.src_id = "T1";
    stockRequest.dst_id = "T2";
    stockRequest.msg_content["item_code"] = "02"; // 사이다 조회
    stockRequest.msg_content["item_num"] = "1";

    std::cout << "수신된 재고 조회 요청: " << stockRequest.src_id
              << " -> " << stockRequest.dst_id
              << ", 음료코드: " << stockRequest.msg_content.at("item_code") << std::endl;
```

```
    // 요청 메시지 검증
    EXPECT_EQ(stockRequest.msg_type, network::Message::Type::REQ_STOCK);
    EXPECT_EQ(stockRequest.src_id, "T1");
    EXPECT_EQ(stockRequest.dst_id, "T2");
    EXPECT_EQ(stockRequest.msg_content.at("item_code"), "02");

    // 재고 확인 처리
    std::string requestedDrinkCode = stockRequest.msg_content.at("item_code");
    auto availability = inventoryService.checkDrinkAvailabilityAndPrice(requestedDrinkCode);

    EXPECT_EQ(requestedDrinkCode, "02");
```

```
✓ 테스트 2 완료: RESP_STOCK 응답 메시지 생성 성공
[       OK ] UC17Test.GenerateStockInquiryResponse (0 ms)
[ RUN      ] UC17Test.HandleOutOfStockInquiry
=== UC17 테스트: 재고 없는 음료 조회 요청 처리 ===
T1 자판기 재고: 콜라(5), 사이다(0), 녹차(10)
재고 조회 요청: T5 -> T1, 사이다 재고 문의
응답 메시지: 사이다 재고 0개 (재고 없음)
✓ 테스트 3 완료: 재고 없는 음료에 대한 응답 처리
[       OK ] UC17Test.HandleOutOfStockInquiry (0 ms)
[ RUN      ] UC17Test.HandleMultipleStockInquiries
=== UC17 테스트: 여러 음료 재고 조회 연속 처리 ===
T2 자판기 재고: 콜라(0), 사이다(12), 캔커피(8), 물(15), 에너지드링크(3)
처리 중: T1에서 음료 02 문의
응답: 02 재고 12개
처리 중: T3에서 음료 08 문의
응답: 08 재고 8개
처리 중: T4에서 음료 01 문의
응답: 01 재고 0개
처리 중: T5에서 음료 09 문의
응답: 09 재고 15개
✓ 테스트 4 완료: 여러 음료에 대한 연속 재고 조회 처리
[       OK ] UC17Test.HandleMultipleStockInquiries (0 ms)
[ RUN      ] UC17Test.HandleInvalidStockInquiry
=== UC17 테스트: 잘못된 재고 조회 요청 처리 ===
T3 자판기에서 잘못된 요청 처리 테스트
잘못된 요청 1 (item_code 누락): 예외 발생 - unordered_map::at: key not found
존재하지 않는 음료 코드 '99': 가격 0원, 재고 0개
올바른 요청: 콜라 재고 0개, 가격 1000원
✓ 테스트 5 완료: 잘못된 요청 거부, 올바른 요청 처리
[       OK ] UC17Test.HandleInvalidStockInquiry (0 ms)
[----------] 5 tests from UC17Test (0 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 5 tests.
```

6. OOAD를 수행한 개인별 소감

김진솔	객체지향을 설계할 때 생기는 절차 플로우를 다루기 위한 아키텍쳐를 설계하는 것이 가장 재미있었다. 머릿속에서 이루어지던 설계 과정에 단계와 기술을 명명한 후 다이어그램으로 나타내는 것이 어려웠지만, 다른 사람과 정확히 일치된 정보를 공유할 수 있다는 점이 매력적이었다. 공을 많이 들인 문서를 따라 구현을 시작했으나, 결국 구현을 따라 문서를 수정하는 나를 발견했다. 문서와 구현을 완벽히 일치시키고 싶었는데 그러지 못해서 아쉬웠고, 수정이 필요없는 설계서를 만들어 보고 싶다. 설계 과정에서 한 사람의 주관은 높은 일관성을, 여러 사람의 협력은 완결성을 가져오는 명확한 장단점이 있음을 생각했다. 이러한 작업이 익숙해질 때쯤 프로젝트가 끝나서 아쉽고 처음부터 제대로 해보고 싶었다.
백현준	c++ 코드도 미흡했었는데 구현하기전에 ooad 설계부터 감이 안와 힘이 들었던 기억이 있었습니다. 객체지향기법에 익숙하지 않았지만 그 기법을 이해하고 실제 프로젝트에 적용한 것은 처음이었고 회사에서 쓰일 수 있다는 것을 미리 경험한 것 같아서 좋은 경험이라는 생각이 들었습니다. 한 단계식 진행할 때마다 점점 감이 와닿았었고 구현 단계에서 설계에서 얼마나 꼼꼼히 작성을 해야 힘이 덜 드는지 몸소 이해가 되었기 때문에 앞으로 구현할 때는 이번 경험을 바탕으로 더 좋은 구현이 되도록 구성을 해야겠다는 생각이 들었습니다.
이현	Use Cass와 sequence diagram을 기반으로 설계부터 구현까지 진행하는 과정을 배웠다. 처음에 Use Case와 Domain Model을 정확하게 정하지 않아서 문제가 생겼다. 문서와 코드의 버전 관리 체계가 미흡한 부분들을 팀원들의 도움을 받고 소통해서