POFS
Vision

1. Introduction

POFS(point of food sale)프로젝트의 vision을 정의한 문서입니다.

2. Positioning

2.1 Problem Statement

The problem of	음식을 주문 하는 것에서부터 음식을 서비스 받을 때까지 시간이 오래 걸리는 것, 여러 종류의 음식을 한 곳에서 주문하지 못하는 것이 문제입니다
affects	Cashier, Manager, CEO, Customer, Cook
the impact of which is	기다리는 시간이 긴 것과 한 곳에서 다양한 종류의 음식을 주문하지 못하는 것에서 고객 불만이 생기고 심한 경우 환불 요청까지 이어질 수도 있습니다.
a successful solution would be	POFS(POS시스템)을 도입하는 것이 성공적인 해결책일 것입니다. POFS(POS시스템)을 도입할 경우, 직원이 직접 구두로 주문내용을 전달하거나 전달과정상 잘못 전달되는 것을 막는 것과 주문처리와 전달에 필요한 시간이 줄어들어서 이전과 비교했을 때 보다 더 많은 주문을 처리하는 것을 기대할 수 있습니다

2.2 Product Position Statement

For	한국의 Food Court 운영 회사
Who	Food Court 내에서 편리한 주문 처리 시스템을 이용하고자 하는
The (product name)	POFS은 POS시스템입니다.
That	여러 가지 음식을 한 곳에서 주문&결제할 수 있는 서비스를 제공합니다.

3. Stakeholder Descriptions

3.1 Stakeholder Summary

Name	Description	Responsibilities
Cashier	이 프로젝트로부터 생성된 결과물(시스템)을 이용해서 고객의 주문을 받고 결제를 해주는 사람입니다.	주문접수, 결제 등 Cashier가 해야 하는 업무에 대해 설명 또는 평가할 책임이 있습니다.
Manager	Food Court에서 발생하는 각종 문제들에 대한 총괄적인 해결을 하는 사람입니다.	일일 마감과 정산 등에 대한 프로젝트의 진행상황을 보고 피드백을 줄 책임이 있습니다.
CEO	이 프로젝트를 발주하는 사람입니다.	프로젝트 비용 승인에 대한 책임을 가지고 있습니다.
Customer	Cashier에게 음식 주문을 한 뒤 결제를 하고 음식을 서비스 받는 사람입니다.	이 시스템에 있는 고객과의 업무처리 기능들에 대한 피드백을 줄 책임이 있습니다.
Cooker	시스템에서 알려주는 주문을 받아서 요리를 하는 사람입니다.	주문할 수 있는 음식의 리스트를 제공해주고 시스템이 조리사들에게 주문을 알려주는 방식에 대한 피드백을 줄 책임이 있습니다.

3.2 User Environment

Cashier는 무조건 1명, 고객도 1명으로 항상 2명의 사람이 주문처리라는 일을 완전히 처리하기 위해서 필요합니다. 이 프로젝트에서는 더치 페이를 허용하지 않는 것으로 정했습니다. 그러므로 일을 완전히 처리하기 위해서는 고정적으로 2명이 필요합니다.

한개의 주문처리를 하는데 걸리는 시간은 1분 30초로 가정하였습니다. 그리고 주문처리과정 중 카드 승인 절차 부분에서 카드 승인 절차가 실패되는 경우 추가적으로 30초의 시간이 더 걸릴 수 있을 거라고 가정하였습니다. 따라서 카드 승인 절차에 의해 주문처리에 소요되는 시간이 변동될 수 있습니다.

요즘 일반적으로 Pos시스템에 쓰이는 시스템 플랫폼은 Windows XP, POSReady7, Windows 10, Linux등이 있습니다. 미래에는 Windows XP는 보안 문제 때문에 사용되지 않고 POSReady7은 지원기간이 2021년에 끝나면서 사용되지 않고 결국 Windows 10, Linux가 미래에 쓰일 시스템 플랫폼이라고 생각합니다.

4. Product Overview

4.1 Needs and Features

Need	Priority	Features	Planned Release
주문 관리	Cashier	현금과 카드를 통한 계산과 관리.	Iteration 1
매출 관리	Manager	일별, 주별, 월별 매출 금액 표시.	Iteration 1
매출 통계	Manager	종류별 음식 매출에 관한 통계.	Iteration 1
메뉴 관리	Manager	음식과 카테고리 정보를 관리.	Iteration 1
주문 알림	Customer	손님에게 음식이 나온 것을 알리는 기능	Iteration 1

5. Other Product Requirements

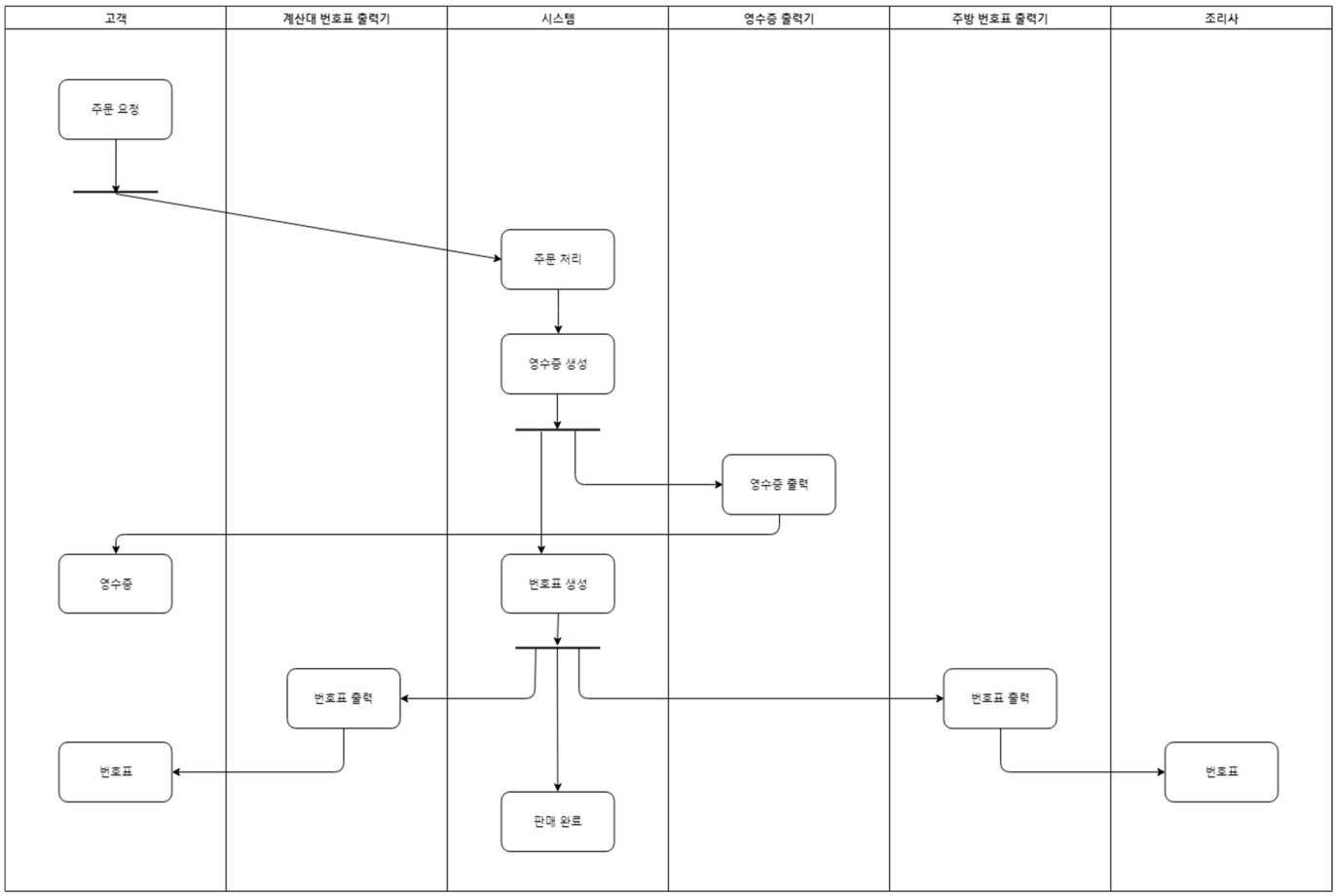
Requirement	Priority	Planned Release
시스템 고장 시 재시작.	높음	Iteration 1
주문 처리 과정이 빨라야함.	높음	Iteration 1
고객이 주문 완료를 쉽게 파악해야 함.	보통	Iteration 1
플랫폼으로 비용이 적은 리눅스 & JVM 사용	보통	Iteration 1
User Manual 및 A/S 시스템 필요	보통	Iteration 1
주문 취소 및 환불이 가능해야함	높음	Iteration 1

음식재고 소진 시 주문중단	높음	Iteration 1
번호표 분실 시 번호표 재발급	보통	Iteration 1
모든 번호를 표시할 수 없음	낮음	Iteration 1
계산대 직원은 고객과 직접적으로 접촉하고 주문처리는 포스 시스템으로 함	높음	Iteration 1
카드와 현금의 결제가 모두 가능해야함	높음	Iteration 1

6. Scope

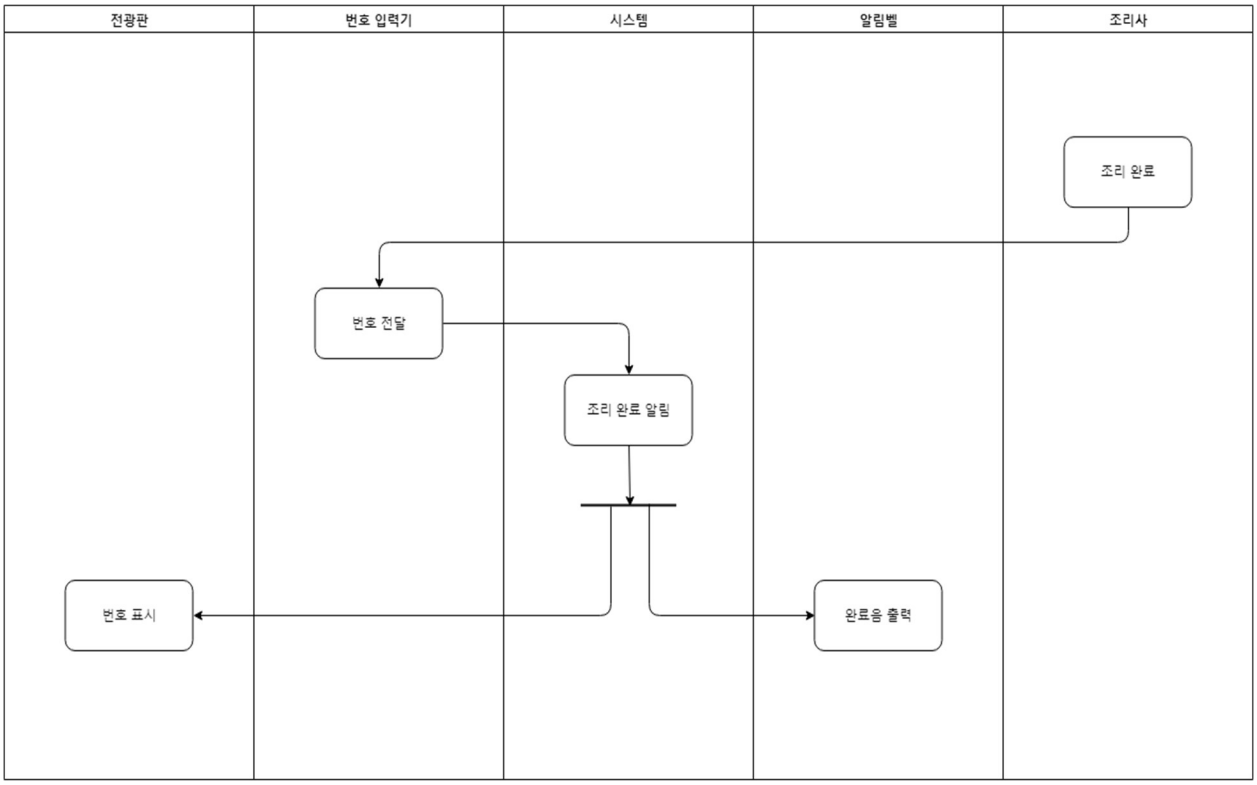
- POFS 는 하드웨어의 인터페이스에 해당하는 번호표와 영수증 기계, 번호판 등을 객체로 대체하여 구현할 예정입니다.
- POFS 의 결제 방식은 카드 결제와 현금 결제 방식을 둘 다 채택하기로 하였습니다.
- POFS 의 FoodCourt 시스템은 최대 15 여개의 주방과 150 여개 정도의 음식 품목을 관리할 수 있게 만들 것입니다.
- POFS 는 하나의 계산대를 통해서 모든 주문을 처리 및 관리할 수 있게 만들 예정입니다.

-주문처리



하나의 주문을 처리할 때 시스템과 연관된 외부 기기 또는 사람들을 표현한 activity diagram입니다.

-조리 완료 알림



음식이 조리 완료 된 후 시스템과 연관된 외부 기기 또는 사람들을 표현한 activity diagram입니다.

POFS
Supporting Requirements Specification

1. Introduction

Use-case에서 설명하지 못한 요구사항들과 제약조건들을 정리한 문서입니다.

2. System wide Functional Requirements

Use-case에 자세히 기술되어 있습니다.

3. System Qualities

3.1 Usability

번호판이 각 주방에 있어야 하고, 30m 정도에서도 식별할 수 있게 커야합니다. 최소 5개의 번호를 보여줄 수 있어야 합니다. 또한, 고객이 음식이 나온 것을 소음이 클때도 명확하게 소리로 인지할 수 있어야 합니다. 출납원이 쓰기에 충분히 시스템이 간편하며 소리 등으로 실수를 방지할 수 있게 만들어져야 합니다.

3.2 Reliability

내부 시스템에 문제가 생겼을 때, 재부팅 후 자동으로 복구할 수 있어야 합니다.

시스템이 갑자기 꺼지는 것을 대비해서 결제내역, 주문내역 등은 하나의 판매가 끝나면 바로 파일에 저장해야 합니다.

푸드 코트의 매출 및 음식 관리 데이터가 프로그램이 꺼져도 보존되도록 DB나 파일 시스템을 활용하여 저장해야 합니다.

3.3 Performance

고객의 만족도를 위해서 주문의 90%를 1분 30초 이내로 처리해야합니다. 또한, 카드 승인 절차가 10초 이내에 이루어 질 수 있도록 해야합니다.

3.4 Supportability

해당 시스템을 사용할 음식점들을 위해서 다양한 음식 메뉴들을 추가하거나 수정할 수 있어야 합니다. 또한, 번호판이나 번호표 출력기, 영수증 출력기 등의 모듈 등이 고장날 시 다른 제품으로 자유롭게 교체할 수 있도록 해야 합니다.

4. Constraints

4.1 Implementation constraints

시스템을 JVM 위에서 이용하는 것이 장기간의 supportability를 제공할 것입니다.

메뉴, 주문 내역, 결제 내역 등을 저장하고 관리하는 기능을 제공할 것입니다.

4.2 Interfaces

Pos기, 영수증 출력기, 번호표 출력기(주방과 계산대에서 쓰는 번호표를 출력해주는 기기입니다.), 주방 번호 입력기(주방에서 번호 호출 전광판으로 번호를 입력하는 입력기입니다.) , 고객 번호 호출 전광판(조리 완료시 고객의 주문번호를 알려주는 전광판입니다.), 알림벨(조리 완료시 주문 번호가 출력된 것을 알려주는 알림벨입니다.)이 필요합니다.

주문처리 activity diagram에 대한 추가설명:

고객이 주문 요청을 할 경우 시스템은 주문 처리를 수행하고 영수증 생성을 수행합니다. 영수증 생성에서 시스템은 영수증을 출력하도록 영수증 출력기에 영수증 출력요청을 보냅니다. 동시에 시스템은 번호표 생성을 수행합니다.

그다음 동시에 시스템은 계산대 번호표 출력기와 주방 번호표 출력기에 번호표 출력 요청을 보내고 판매 완료를 수행합니다.

시스템에 계산대 번호표 출력기와 영수증 출력기는 각각 1대만 연결되어있고 주방 번호표 출력기는 주방의 개수 만큼 확장해서 연결 가능하게 할 계획입니다.

조리 완료 알림 activity diagram에 대한 추가설명:

조리사가 번호 입력기를 통해 조리 완료된 음식의 번호를 시스템에게 알리면 시스템이 각 주방 밖에 있는 전광판에 번호를 표시하고 완료음을 냅니다.

시스템에는 주방의 개수 만큼의 전광판이 연결되어 있고 알림벨은 1개만 연결 가능하게 할 계획입니다.

5. Business Rulesstraints

5.1 부가가치세법

실제 판매금액의 10%를 부가가치세로 국가에 납세해야하는 것에 관한 법입니다. 이 프로젝트에서는 결제금액 산정 또는 결제요청을 카드사로 보낼때 고려해야하는 법입니다.

5.2 전자금융거래법

전자금융거래에 관한 내용을 다루는 법입니다. 이 프로젝트에서 Cashier가 고객으로부터 카드를 받고 결제를 할 때 Food Court운영 회사와 고객간에 전자금융거래가 이루어지기 때문에 고려해야하는 법입니다.

POFS

Glossary

Term	Definition and Information
번호 입출력기	주방에 있는 기기로서 조리 완료된 번호를 시스템에게 알려주는 역할을 하고 계산대로부터 주문 정보를 받아서 주문정보를 출력해주는 기기입니다. 주문출력 방식은 대부분의 출력기가 채택해서 쓰고있는 감열식 프린팅 방식입니다.
영수증 식별 번호	영수증 속에 기입되어져있는 주문정보 중 하나로 시스템속 주문기록 정보와 영수증을 매칭시킬때 이용됩니다.
카드 승인 요청	판매자에게 지불하거나 지불을 보증하는 외부 지불 인증 서비스에 의한 검증 요청
판매 수량	모든 품목을 포함한 수량을 말합니다.

User Defined Datatype Name	Member	Definition and Information
Address	administrative_district	행정구역을 나타내는 변수이고 type은 String입니다. ex)경기도 수원시 영통구
	street_name	도로명주소에서 도로명을 나타내는 변수이고 type은 String입니다.

		ex)월드컵로
	building_number	도로명주소에서 도로명뒤에 나오는 건물번호를 나타내는 변수이고 type은 Integer입니다. ex)206
	detailed_address	도로명주소에서 건물번호 뒤에 나오는 상세주소(아파트의 경우 동호수)를 나타내고 type은 String입니다. ex)아주대학교

POFS
Use-Case: 주문 처리

1 Brief Description

Cashier가 고객의 주문을 처리하는 과정입니다.

2 Actor Brief Descriptions

2.1 Cashier: 고객의 주문을 처리하는 Actor 입니다.

2.2 Manager: 주문 처리 과정중 발생하는 예외 상황들을 처리해주는 Actor 입니다.

3 Preconditions

계산대에 Pos기와 Pos시스템이 있어야되고 각 주방의 전광판과 번호 입력기에 시스템이 연결되어 있어야 합니다.

4 Basic Flow of Events

1. 고객이 계산대로 온다.
2. Cashier 가 새로운 판매를 시작한다.
3. 고객이 cashier 에게 주문할 음식을 말한다.
4. Cashier 가 주문 받은 음식을 입력한다.
5. 시스템이 고객이 주문한 음식을 기록하고 음식의 가격과 총 가격을 보여준다.

   Cashier가 고객의 주문이 끝날 때까지 3, 4, 5를 반복한다.
6. Cashier 가 고객에게 총 가격을 말해주고 결제를 요청한다.
7. 고객이 현금을 지불한다.
8. Cashier 가 받은 금액을 입력한다.
9. 시스템이 거슬러줄 금액을 보여준다.
10. Cashier 가 고객에게 거스름돈을 준다.
11. 시스템이 판매를 기록하고 매출에 반영한다.
12. 시스템이 영수증을 출력한다.

13. 시스템이 주방과 계산대에 번호표를 출력한다.

14. 고객이 번호표와 영수증을 받고 기다린다.

15. Cashier 가 판매 완료 처리를 한다.

5 Alternative Flows

1a. 고객이 Cashier에게 번호표를 재발급해달라고 요청한다. :

1. Cashier가 고객에게 영수증을 보여달라고 요청한다.
2. 고객이 영수증을 Cashier에게 보여준다.
3. Cashier는 영수증의 식별 번호를 보고 시스템에 입력한다.

3a. 시스템이 식별 번호에 맞는 주문 정보를 찾을 수 없는 경우 :

1. 주문처리 use-case가 끝난다.

4. 시스템이 식별 번호와 일치하는 주문에 대한 정보를 보여준다.
5. main시나리오의 Step 12로 넘어간다.

1b. 고객이 전에 했던 주문을 취소해달라고 한다. :

1. Cashier가 고객에게 영수증을 보여달라고 요청한다.
2. 고객이 영수증을 Cashier에게 보여준다.
3. Cashier는 영수증의 식별 번호를 보고 시스템에 입력한다.

3a. 시스템이 식별 번호에 맞는 주문 정보를 찾을 수 없는 경우 :

1. 주문처리 use-case가 끝난다.

4. 시스템이 식별 번호와 일치하는 주문에 대한 정보를 보여준다.
5. Cashier가 주문 정보에 나와있는 금액만큼 고객에게 환불해준다.

5a. 계산대에 환불해줄 현금이 없거나 부족한 경우 :

1. Cashier가 매니저에게 현금이 부족하다고 말한다.
2. 매니저가 은행에서 현금을 가져온다.
3. Cashier가 주문 정보에 나와있는 금액만큼 고객에게 환불해준다.

6. Cashier가 해당 주문을 취소한다.
7. 시스템이 주방에 주문 취소 정보를 전달한다.
8. 주문처리 use-case가 끝난다.

1c. 매니저가 Cashier에게 재고 없는 음식들을 알려준다. :

1. Cashier가 재고 없는 음식들을 시스템에게 알려준다.
2. 시스템이 음식에 재고가 없음을 기록한다.
3. 주문처리 use-case가 끝난다.

3a. 고객이 자신이 현재 주문한 음식들 중 특정 음식의 주문을 취소해달라고 한다. :

1. Cashier가 주문 리스트에서 해당 음식을 취소한다.
2. 시스템이 취소한 음식의 가격을 기존의 총 가격에서 빼고 총 가격을 보여준다.
3. main시나리오의 Step 3에서 main시나리오를 이어서 진행한다.

4a. 시스템이 Cashier에게 입력한 음식의 재고가 없다고 알려줄 경우 :

1. Cashier가 고객에게 해당 음식이 매진되었다고 말한다.
2. main시나리오의 Step 3에서 main시나리오를 이어서 진행한다.

3-6a. 고객이 현재의 주문 전체를 취소해달라고 한다. :

1. Cashier가 현재 판매를 종료한다.
2. 주문처리 use-case가 끝난다.

7a. 고객이 카드를 준다.

1. Cashier 가 카드 결제 버튼을 누른다.
2. Cashier 가 카드를 POS 기에 긁는다.
3. 시스템이 카드사에 카드 승인 요청을 한다.

    3a. 카드의 승인이 거절된다.

        1. 메인 시나리오의 step 7번으로 시작한다.

4. 카드사에서 시스템에게 승인을 한다.
5. 메인 시나리오의 step 11 번부터 시작한다.

7-10a. 고객이 현재의 주문 전체를 취소해달라고 한다. :

1. Cashier가 주문 정보를 확인한다.
2. Cashier가 주문 정보에 나와있는 금액만큼 고객에게 환불해준다.
3. Cashier가 현재 판매를 종료한다.
4. 주문처리 use-case가 끝난다.

11-14a. 고객이 현재의 주문 전체를 취소해달라고 한다. :

1. Cashier가 주문 정보를 확인한다.
2. Cashier가 주문 정보에 나와 있는 금액만큼 고객에게 환불해준다.
3. 시스템이 주방에 주문 취소 정보를 전달해 준다.
4. Cashier가 현재 판매를 종료한다.
5. 주문처리 use-case가 끝난다.

2-6a. 매니저가 Cashier에게 재고 없는 음식들을 알려준다. :

1. Cashier가 현재 주문에 해당사항이 없는지 확인한다.

1a. 현재 주문에 해당사항이 없는 경우 :

1. Cashier가 재고 없는 음식들을 시스템에게 알려준다.
2. main시나리오로 되돌아가서 나머지 main 시나리오의 Step들을 진행한다.

1b. 현재 주문에 해당사항이 있는 경우 :

1. Cashier가 재고 없는 음식들을 시스템에게 알려준다.
2. 시스템이 현재 주문리스트에서 재고가 없어진 음식들을 제거하고 총 가격을 수정해서 보여준다.
3. Cashier가 고객에게 주문한 음식 중 재고 부족으로 주문 할 수 없는 음식들을 알려준다.
4. main시나리오의 Step 3에서 main시나리오를 이어서 진행한다.

7-13b. 매니저가 Cashier에게 재고 없는 음식을 알려준다. :

1. Cashier가 현재 주문에 해당사항이 없는지 확인한다.

1a. 현재 주문에 해당사항이 없는 경우 :

1. Cashier가 재고 없는 음식들을 시스템에게 알려준다.
2. 시스템이 음식에 재고가 없음을 기록한다.
3. main시나리오로 되돌아가서 나머지 main 시나리오의 Step들을 진행한다.

1b. 현재 주문에 해당사항이 있는 경우 :

1. Cashier가 재고 없는 음식들을 시스템에게 알려준다.
2. 시스템이 음식에 재고가 없음을 등록한다.

3. Cashier가 받은 금액을 고객에게 되돌려 준다.

4. Cashier가 현재 판매를 취소한다.

5. main시나리오의 Step 2에서 main시나리오를 이어서 진행한다.

14a. 매니저가 Cashier에게 재고 없는 음식을 알려준다. :

1. Cashier가 고객이 계산대 앞에 있는지 확인한다.

1a. 고객이 계산대 앞에 있는 경우 :

1. Cashier가 고객으로부터 영수증을 받는다.

2. Cashier가 영수증의 식별번호를 시스템에 입력한다.

2a. 시스템이 식별 번호에 맞는 주문 정보를 찾을 수 없는 경우 :

1. 주문처리 use-case가 끝난다.

3. 시스템은 해당 주문을 보여준다.

4. Cashier가 주문 정보에 나와있는 금액을 고객에게 돌려준다.

5. Cashier가 해당 주문을 취소한다.

6. 시스템이 주방에 주문 취소 정보를 전달해 준다.

7. main시나리오의 Step 2에서 main시나리오를 이어서 진행한다.

1b. 고객이 계산대 앞에 없는 경우 :

1. 매니저가 고객을 찾아서 계산대로 데려온다.

2. Cashier가 고객으로부터 영수증을 받는다.

3. Cashier가 영수증의 식별번호를 시스템에 입력한다.

3a. 시스템이 식별 번호에 맞는 주문 정보를 찾을 수 없는 경우 :

1. 주문처리 use-case가 끝난다.

4. 시스템은 해당 주문을 보여준다.

5. Cashier가 주문 정보에 나와있는 금액을 고객에게 돌려준다.

6. Cashier가 해당 주문을 취소한다.

7. 시스템이 주방에 주문 취소 정보를 전달해 준다.
8. main시나리오의 Step 2에서 main시나리오를 이어서 진행한다.

10a. 계산대에 거스름돈이 없는 경우 :

1. cashier 가 주문을 멈춘다.
2. chier 가 매니저에게 거스름돈이 없음을 알린다.
3. 매니저가 은행으로 가서 잔돈으로 바꾼다.
4. 매니저 가 계산대로 거스름돈을 가지고 돌아온다.
5. cashier 가 시스템에서 최종금액을 확인하고 거스름돈을 준다.
6. main 시나리오의 step11 에서 main 시나리오를 이어서 진행한다.

12-13a. 번호표와 영수증이 출력되지 않는 경우 :

1. 해당출력기기를 열어 용지가 있는지 확인한다.

1a. 용지가 있는 경우

1. 해당 출력기기를 재부팅한다.
2. 해당 주문의 영수증을 다시 출력한다.

2a.영수증이 출력되지 않는 경우 :

1.새로운 출력기기로 교체한다.

2.해당 주문의 영수증을 다시 출력한다.

3.main시나리오로 돌아가서 나머지 main시나리오의 step들을 진행한다.

3. main 시나리오로 돌아가서 나머지 main 시나리오의 step 들을 진행한다.

1b. 용지가 없는경우

1. 해당 출력기기에 cashier 가 용지를 넣는다.
2. 해당 출력기기를 재부팅한다.
3. cashier 가 해당주문의 영수증을 다시 출력한다.
4. main 시나리오로 되돌아가서 나머지 main 시나리오의 Step 들을 진행한다.

6 Post-conditions

6.1 Successful Completion

고객은 영수증과 번호표를 받고 음식이 나올때까지 기다릴 수 있습니다.

6.2 Failure Condition

주문처리에 실패한 원인을 시스템이 기록해야 합니다.

7 Special Requirements

카드 결제에서 결제에 실패한 경우 그 원인들을 시스템이 log로 남겨야 됩니다.

POFS
Use-Case: 메뉴 관리

1 Brief Description

Manager가 시스템에 음식과 카테고리, 주방 정보를 관리하는 과정입니다.

2 Actor Brief Descriptions

2.1 Manager: 시스템에 음식과 카테고리, 주방 정보를 등록하는 Actor 입니다.

3 Preconditions

계산대에 Pos기와 Pos시스템이 있어야되고 시스템이 시작한 상태여야 합니다.

4 Basic Flow of Events

1. Manager 가 시스템에게 주방 및 음식 수정을 요청한다.
2. 시스템이 Manager 에게 주방과 음식의 리스트를 보여준다.
3. Manager 가 추가할 음식의 이름과 가격을 입력한다.
4. Manager 가 추가할 음식이 속하는 주방 이름을 입력한다.
5. 시스템이 입력된 정보를 통해서 주방과 음식의 리스트를 업데이트한다.
6. 시스템이 Manager 에게 업데이트된 주방과 음식의 리스트를 보여준다.

   Manager가 step 3-6을 원하는 만큼 반복한다.
7. 시스템이 변경된 사항을 저장한다.

5 Alternative Flows

3a. Manager가 주방 정보와 주방에서 다루는 음식 카테고리를 추가하려고 한다:

1. Manager 가 주방과 카테고리의 정보를 입력한다.

   1a: 주방과 카테고리의 정보가 너무 많은 경우:

   1. 시스템이 더 이상 정보를 추가할 수 없음을 알린다.
   2. main 시나리오의 step 6 으로 돌아간다.
2. main 시나리오의 step 5 로 돌아간다.

3b. Manager가 음식을 삭제하려고 한다:

1. Manager 가 삭제할 음식을 선택한다.
2. 시스템이 선택된 음식을 삭제한다.

   2a: 삭제할 음식이 존재하지 않는 경우:

   1. 시스템이 Manager 에게 선택한 항목이 존재하지 않음을 알려준다.
   2. main 시나리오의 step 6 으로 돌아간다.
3. main 시나리오의 step 6 으로 돌아간다.

3c. Manager가 주방 정보나 카테고리를 수정하려고 한다:

1. Manager 가 변경할 주방이나 카테고리를 선택한다.
2. Manager 가 변경 사항을 입력한다.

   2a. 선택한 주방이나 카테고리가 존재하지 않는 경우:

   1. 시스템이 Manager 에게 선택한 항목이 존재하지 않음을 알려준다.
   2. main 시나리오의 step 6 으로 돌아간다.
3. main 시나리오의 step 5 로 돌아간다.

3d. Manager가 주방과 카테고리를 삭제하려고 한다:

1. Manager 가 삭제할 주방과 카테고리를 선택한다.
2. 시스템이 Manager 에게 삭제할 주방과 카테고리를 확인시킨다.

   2a. 선택한 주방과 카테고리가 존재하지 않는 경우:

   1. 시스템이 Manager 에게 선택한 항목이 존재하지 않음을 알려준다.
   2. main 시나리오의 step 6 으로 돌아간다.
3. Manager 가 시스템에게 확인했음을 알린다.

   3a. Manager가 시스템에게 잘못된 결정임을 알리는 경우:

   1. main 시나리오의 step 6 으로 돌아간다.
4. 시스템이 주방 정보와 카테고리에 속한 모든 음식을 삭제한다.
5. 시스템이 주방 정보와 카테고리를 삭제한다.
6. main 시나리오의 step 6 으로 돌아간다.

5a. 음식 정보가 너무 많아서 더 이상 추가할 수 없는 경우:

1. 시스템이 Manager 에게 음식을 더 이상 추가할 수 없음을 알린다.
2. main 시나리오의 step 6 으로 돌아간다.

5b. 입력한 주방 이름이 존재하지 않는 경우:

1. 시스템이 Manager 에게 선택한 항목이 존재하지 않음을 알린다.
2. main 시나리오의 step 6 으로 돌아간다.

7a. 변경된 사항을 저장할 수 없는 경우:

1. 시스템이 Manager 에게 변경된 사항을 저장하는데 실패했음을 알린다.
2. 주방 및 음식 관리 Use-case 를 종료한다.

6 Post-conditions

6.1 Successful Completion

FoodCourt는 음식과 주방 정보, 카테고리를 입맛에 맞춰서 원하는 대로 관리할 수 있습니다.

6.2 Failure Condition

처리하는 과정에서 실패한 원인을 시스템이 기록해야 합니다.

7 Special Requirements

주방 및 음식의 리스트가 안정적으로 저장될 수 있어야 합니다.

POFS

Use-Case: 매출 관리

1 Brief Description

Manager가 food court의 총 매출을 관리하고 처리하는 과정입니다.

2 Actor Brief Descriptions

2.1 Manager : food court 의 영업을 종료시키고 총 매출을 관리하는 Actor 입니다.

3 Preconditions

판매가 시스템에 기록돼 있어야 합니다.

4 Basic Flow of Events

1. 매니저가 시스템에게 카드 일매출 정보를 요청한다.
2. 시스템이 카드 일매출을 보여준다.
3. 매니저가 시스템에게 현금 일매출 정보를 요청한다.
4. 시스템이 현금 일매출 정보를 보여준다.
5. 매니저가 현금과 카드 일매출 정보의 합계를 요청한다.
6. 시스템이 현금과 카드 일매출 정보의 합계를 보여준다.
7. 매니저가 영업마감을 시스템에 입력한다.
8. 시스템이 마감정산을 등록한다.

5 Alternative Flows

*a. 언제든지, 시스템이 실패한 경우.

1. 매니저가 시스템을 재시작하고 복구를 요청한다.
2. 시스템이 최근의 정상적인 상태로 복구한다.

   2a. 시스템이 복구에 실패한 경우:

   1. 시스템이 매니저에게 복구를 실패했음을 알린다.
   2. 시스템이 기록을 초기 상태로 되돌린다.
   3. 메인 시나리오의 step 1 로 돌아간다.

6 Post-conditions

6.1 Successful Completion

매출 관리에 성공하면 시스템을 종료시킬 수 있습니다.

6.2 Failure Condition

매출 관리에 실패하면 매니저가 문제 사항을 찾고 이를 매출장부에 기록해야 합니다.

7 Special Requirements

매출 관리를 위해 시스템이 모든 판매에 대한 날짜와 시간을 기록하고 있어야합니다.

POFS
Use-Case: 조리 완료 알림

1 Brief Description

조리 완료 알림이 고객에게 전달되는 과정입니다.

2 Actor Brief Descriptions

2.1 Cook: 조리 완료를 시스템에게 알리는 Actor 입니다.

2.2 Manager: 조리 완료를 시스템에 알리고 시스템이 다른 외부장치들과 잘 동작하는지 확인 후 조치를 취하는 Actor 입니다.

2.3 Cashier: 조리 완료 정보가 자동으로 시스템에게 전달되지 않는 경우 시스템에게 직접 전달해주는 Actor 입니다.

3 Preconditions

포스 시스템에 각 주방의 번호 입출력기와 전광판이 연결되어 있어야 합니다.

4 Basic Flow of Events

1. 주방에서 Cook 가 음식 조리를 완료한다.
2. Cook 가 그 음식에 해당하는 번호를 주방에 있는 번호 입력기에 입력한다.
3. 시스템이 번호 입력기에서 입력받은 번호를 전광판에 표시해 준다.
4. 시스템이 번호가 출력될 때 마다 주문완료음을 낸다.
5. 고객이 번호를 보고 음식을 가지고 간다.

5 Alternative Flows

2a. Cook가 번호를 잘못 입력한 경우 :

1. Cook 가 번호 입출력기를 통해 시스템에 조리 완료 처리 취소요청을 보낸다.
   1a. Cook가 시스템에 취소요청한 번호가 대기목록에 없는 번호인 경우:
      1. 조리 완료 알림 use-case 가 끝난다.
2. 시스템이 전광판에서 취소요청한 번호를 삭제한다.

3. 조리 완료 알림 use-case 가 끝난다.

2b. 번호 입력기가 동작하지 않는 경우:

1. Cook 가 번호 입출력기를 재부팅시킨다.

1a. 번호 입력기가 재부팅 후 정상작동 하는 경우 :

1. main 시나리오의 Step 2 에서 main 시나리오를 이어서 진행한다.

1b. 번호 입력기가 재부팅 후에도 정상작동 하지 않는 경우 :

1. Cook 가 매니저에게 조리 완료된 번호를 말한다.
2. 매니저는 Cashier 에게 직접 조리 완료된 번호를 말해준다.
3. Cashier 는 시스템에게 조리 완료된 번호를 말해준다.

3a. 전광판에 번호가 출력되지 않는 경우 :

1. Cook 가 매니저에게 전광판 이상문제를 말한다.
2. 매니저는 적합한 조치를 취한다.

4a. 주문완료음이 나오지 않는 경우 :

1. Cook 가 매니저에게 전광판 이상문제를 말한다.
2. 매니저는 적합한 조치를 취한다.

6 Post-conditions

6.1 Successful Completion

이 use-case가 성공적으로 끝나는 경우 고객은 조리 완료된 음식을 서비스 받을 수 있습니다.

6.2 Failure Condition

이 use-case가 성공적으로 끝나지 못한 경우 매니저는 실패 원인을 찾아야 합니다.

7 Special Requirements

주방에서의 조리 완료 처리에 대한 취소를 요청할 때 조리 완료 처리 요청을 할 때 시스템이 날짜, 시간 등을 포함하는 log를 기록해야합니다.

Use-Case Model Overview

1 Introduction

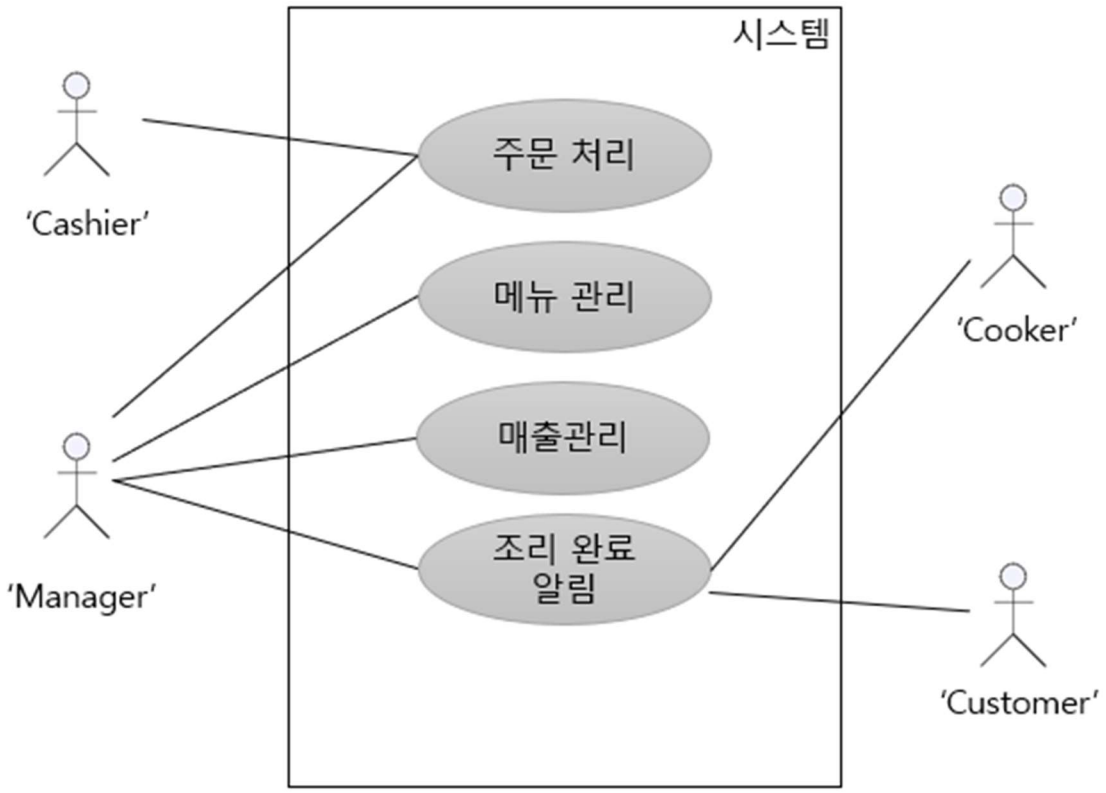
Use case들을 간략하게 설명하고, Use-Case diagram을 그린 문서입니다.

2 Overview

해당 시스템은 주문 처리와 전달에 필요한 시간과 착오를 줄이고, 더 많은 주문을 처리하는데 목표를 가지고 만들어진 시스템입니다. 또한, 이 시스템은 푸드 코트의 Cashier의 수를 줄이고, 주문 착오를 덜어 소비자의 컴플레인을 줄일 것을 기대할 수 있습니다.

3 Use-Case Diagram

아래 그림은 Use-Case 다이어그램을 보여줍니다.



Use-Case Diagram

4 Actors

4.1 Cashier

시스템을 이용해서 고객의 주문을 받고 결제를 해주는 액터입니다.

4.2 Manager

Food Court에서 발생하는 각종 문제에 대한 총괄적인 해결을 하는 액터입니다.

4.3 Cooker

시스템에서 알려주는 주문을 받아서 요리를 하고, 시스템에 완료한 요리 번호를 입력하는 액터입니다.

4.4 Customer

Cashier에게 음식 주문을 한 뒤, 결제를 하고 음식 서비스를 받는 사람입니다.

5 Use Cases

5.1.1 주문 처리

이 Use Case는 Pos기를 이용해서 계산을 처리하는 기능입니다.

5.1.2 메뉴 관리

이 Use Case는 주방과 음식, 카테고리 정보들을 관리하는 기능입니다.

5.1.3 매출 관리

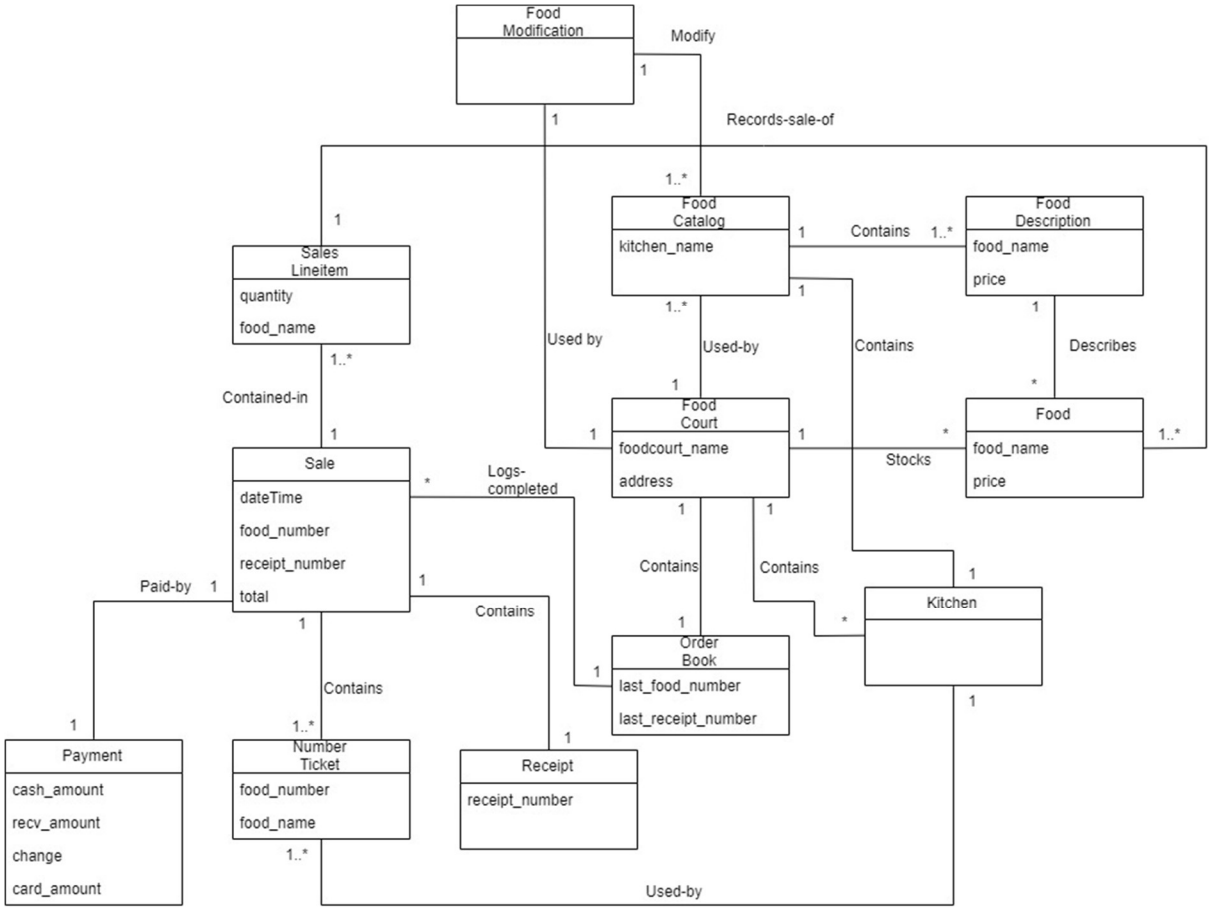
이 Use case는 일별, 월별 매출을 관리하는 기능입니다.

5.1.4 조리 완료 알림

이 Use case는 주방에서 customer에게 조리 완료를 알려주는 기능입니다.

POFS

Domain Model



1. Scope

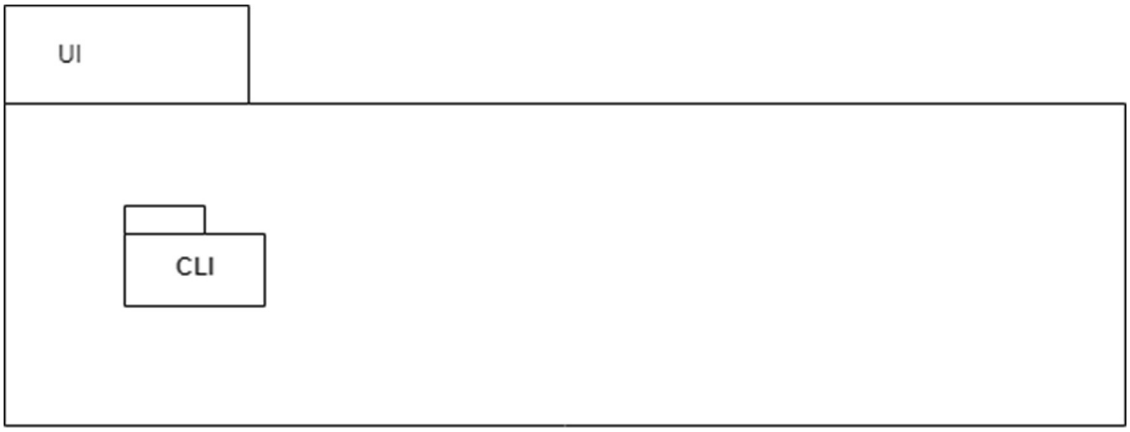
주문처리 use case, 메뉴 관리 use case, 매출 관리 use case, 조리 완료 알림 use case에서 명사나 명사구를 뽑아서 class로 정하고 그 후 use case에서 뽑아낸 개념을 attribute로 결정하고 정해진 class들의 association을 정하였습니다.

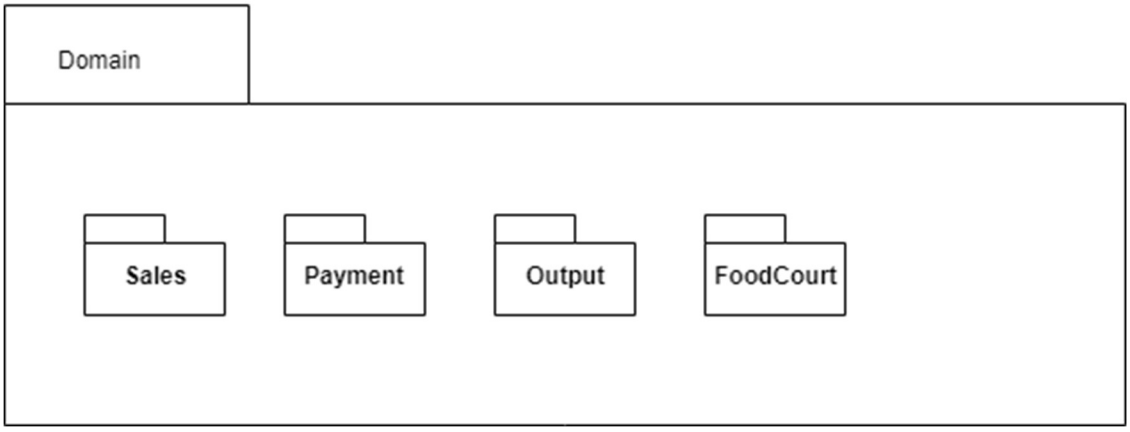
즉 이 Domain Model에서 구현한 범위는 주문처리 use case, 메뉴 관리 use case, 매출 관리 use case, 조리 완료 알림 use case 이 4가지 use case들 입니다.

POFS

Logical architecture

1. **Layered logical architecture**





POFS

Iteration Plan

1. Key milestones

Milestone	Date
Iteration start	5/3
vision에 대한 stakeholder들의 의견확인	5/7
주요 위험요소를 설정하였는지 확인	5/18
vision 및 requirement의 안정성 확인	5/18
construction iteration의 계획이 자세히 정해 졌는지 확인	5/27
Iteration stop	5/31

2. High-level objectives

카드 유효성 문제들이 계산대에서 발생합니다.

3. Work Item assignments

Name or key words of description	Priority	Size estimate (points)	Assigned to (name)	Estimate of hours remaining
Support simple pos system	1	7	조민제, 이예빈, 정안지, 천윤서	44
Do the design			조민제, 이예빈, 정안지, 천윤서	8
계산대에서 사용하는 부분 구현 및 테스트			조민제, 이예빈	15
주방에서 사용하는 부분 구현 및 테스트			정안지, 천윤서	8

4. Issues

- 각자의 일정이 다른데 시간을 어떻게 맞출 것인가?
- 각자의 개발 환경을 어떻게 통일 할 것인가?
- 소스 코드를 작성할 때 네이밍 규칙을 어떻게 통일할 것인가?
- 어떤 문서 규격과 협업을 위한 클라우드를 쓸 것인가?

5. Evaluation criteria

- **95%의 시스템 레벨 테스트를 통과한다.**
- **Iteration 에서 제공된 문서들에 대해서 팀원 모두에게 호의적인 반응을 얻는다.**
- **기술적인 데모에 대해서 팀원 모두에게 호의적인 반응을 얻는다.**

6. Assessment

Assessment target	주문 처리 시나리오에서 제기된 카드 유효성 문제를 해결해라.
Assessment date	2019/05/17
Participants	팀원 전체
Project status	Yellow

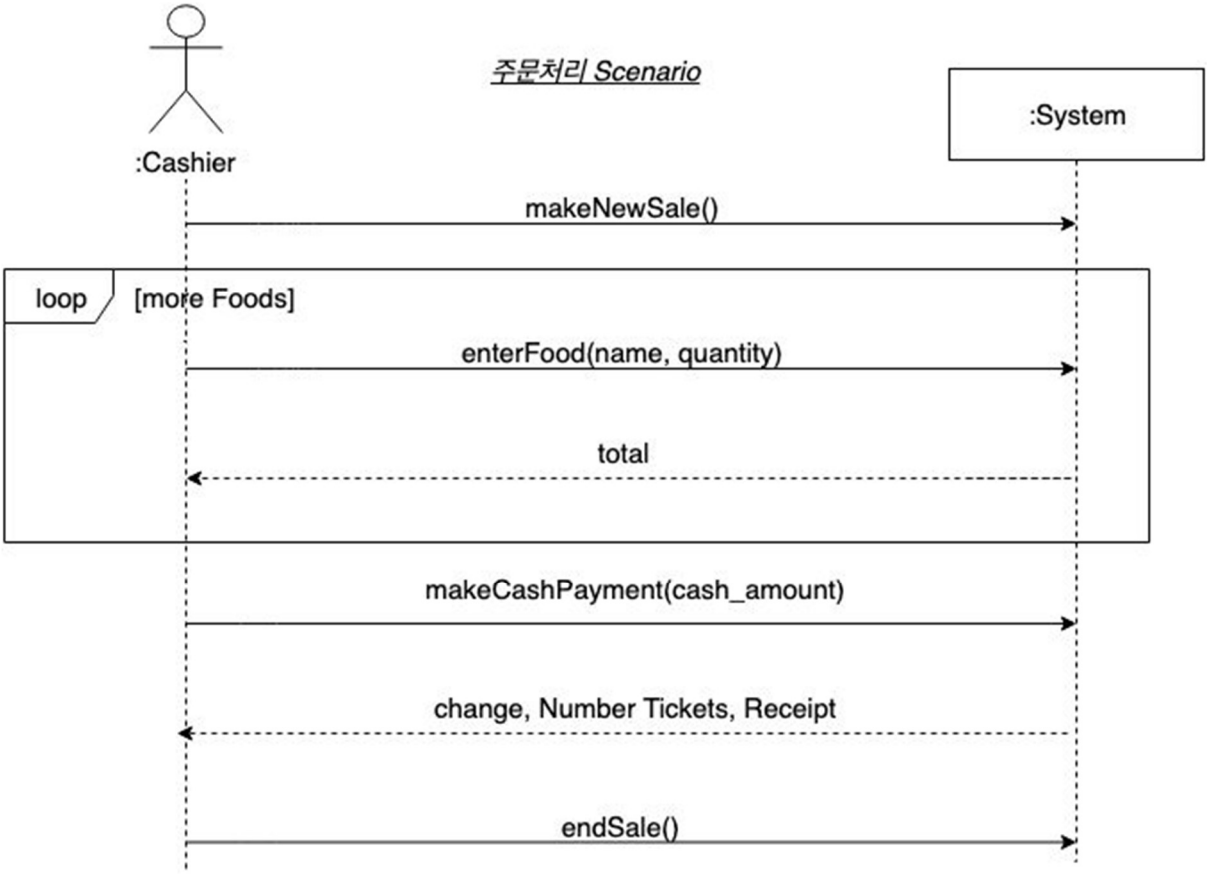
Assessment target	Food catalog와 Kitchen, Food Description간의 관계를 정립하라.
Assessment date	2019/05/09
Participants	팀원 전체
Project status	Green

- **Other concerns and deviations**

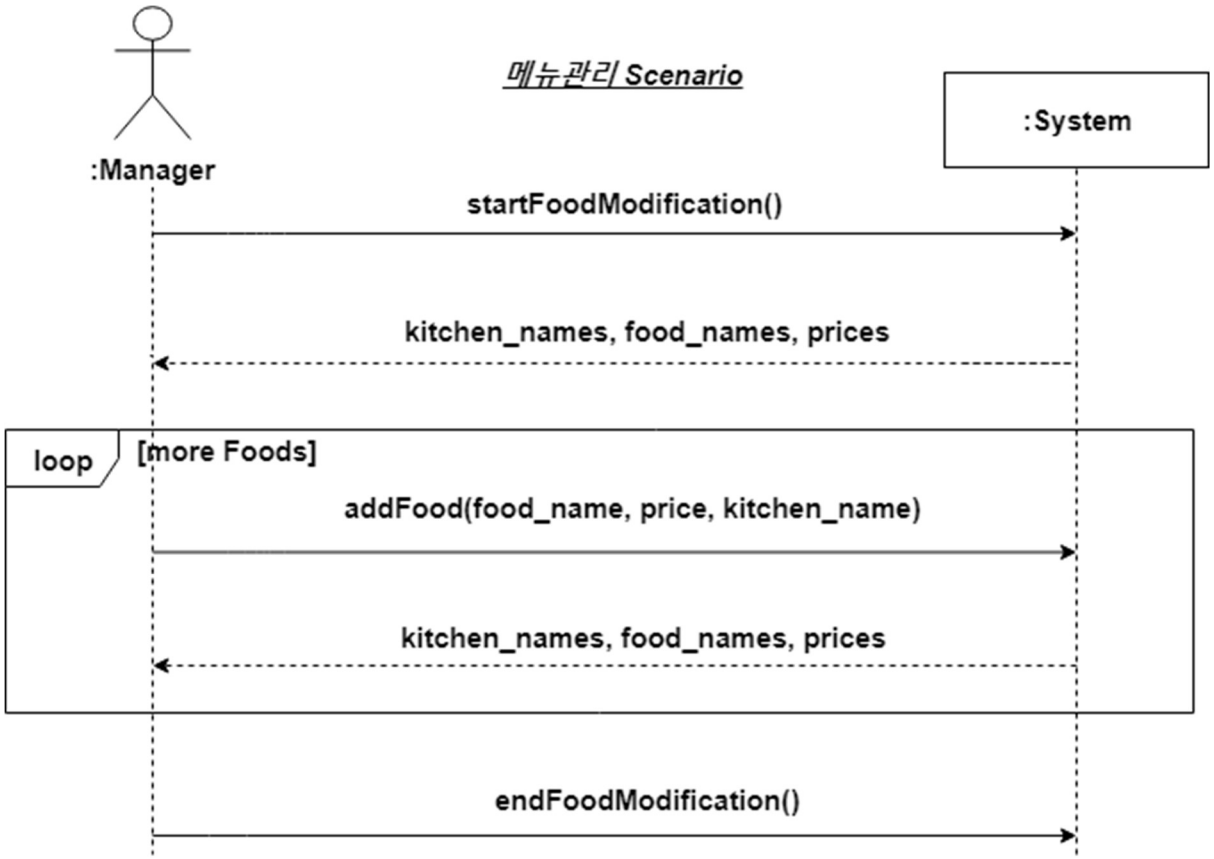
화요일 PM 03:00는 팀원 중 한명이 바로 다음 시간에 퀴즈가 있기 때문에 다른 일정을 찾아보아야 한다.

System Sequence Diagrams

SSD for 주문처리



SSD for 메뉴관리


메뉴관리 Scenario
:Manager
:System
startFoodModification()
kitchen_names, food_names, prices
loop
[more Foods]
addFood(food_name, price, kitchen_name)
kitchen_names, food_names, prices
endFoodModification()

POFS

Operation Contracts

Operation: makeNewSale()

Cross references: 주문처리

Preconditions: 없음

Postconditions:

- Sale 의 객체 s 가 생성되었다.
- s 가 현재의 OrderBook 과 association 을 갖게되었다.
- s 의 속성들이 초기화되었다.

Operation: enterFood(name: String, quantity: Integer)

Cross references: 주문처리

Preconditions: 처리중인 Sale이 있어야 한다.

Postconditions:

- SalesLineitem 의 객체 sli 가 생성되었다.
- sli 는 현재 Sale 과 association 을 갖게 되었다.
- sli.quantity 는 quantity 의 값을 가지게 되었다.
- sli 는 name 을 기반으로 FoodDescription 과 association 을 갖게 되었다.

Opeation: makeCashPayment(cash_amount: Integer, recv_amount: Integer)

Cross references: 주문처리

Preconditions: 처리중인 Sale이 있어야 한다.

Postconditions:

- Payment 의 instance p 가 생성되었다.

- p.cash_amount 는 amount 의 값을 가지게 되었다.
- p.change 는 change 의 값을 가지게 되었다.
- p 는 현재 Sale 과 association 을 갖게 되었다.
- 현재 Sale 은 OrderBook 과 association 을 갖게 되었다.

Operation: endSale()

Cross references: 주문처리

Preconditions: Payment가 생성완료되고, 처리 중인 Sale이 있어야한다.

Postconditions:

- 현재 Sale 을 통해서 NumberTicket 의 instance nt 가 생성되었다.
- 현재 Sale 을 통해서 Receipt 의 instance r 이 생성되었다.
- Sale 의 완료 상태를 설정하였다.

Operation: startFoodModification()

Cross references: 메뉴관리

Preconditions: 생성완료된 foodcourt가 있어야한다.

Postconditions:

- FoodModification 의 객체 f 가 생성되었다.
- f 가 현재 FoodCourt 와 association 을 갖게 되었다.
- f 가 현재 FoodCatalog 와 association 을 갖게 되었다.

Operation: addFood(food_name: String, price: Integer, kitchen_name: String)

Cross references: 메뉴관리

Preconditions: 처리중인 FoodModification이 있어야 한다. 입력된 kitchen_name을 attribute로 가지는 FoodCatalog 객체가 있어야 한다.

Postconditions:

- FoodDescription fd 가 생성되었다.
- fd.food_name 은 food_name 값을 가지게 되었다.
- fd.price 는 price 값을 가지게 되었다.
- fd 가 kitchen_name 을 기반으로 FoodCatalog 와 association 을 가지게 되었다.
-

Operation: endFoodModification()

Cross references: 메뉴관리

Preconditions: 처리중인 FoodModification이 있어야 한다.

Postconditions:

- instance 나 association 의 변화가 없었다.

Interaction Diagrams

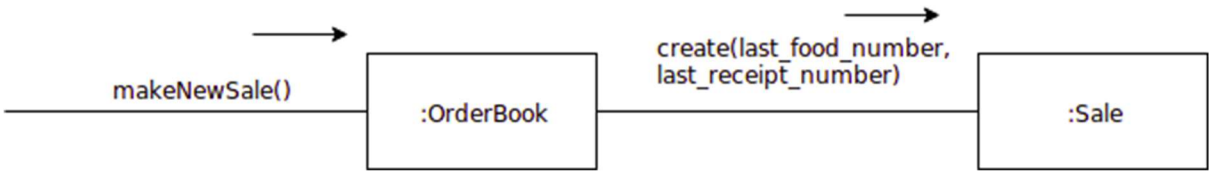
-POFS-

1. Scope

이번 Interaction Diagram들은 주문 처리와 메뉴 관리의 Use case를 구현할 때의 동작을 표현하기 위해서 작성되었습니다.

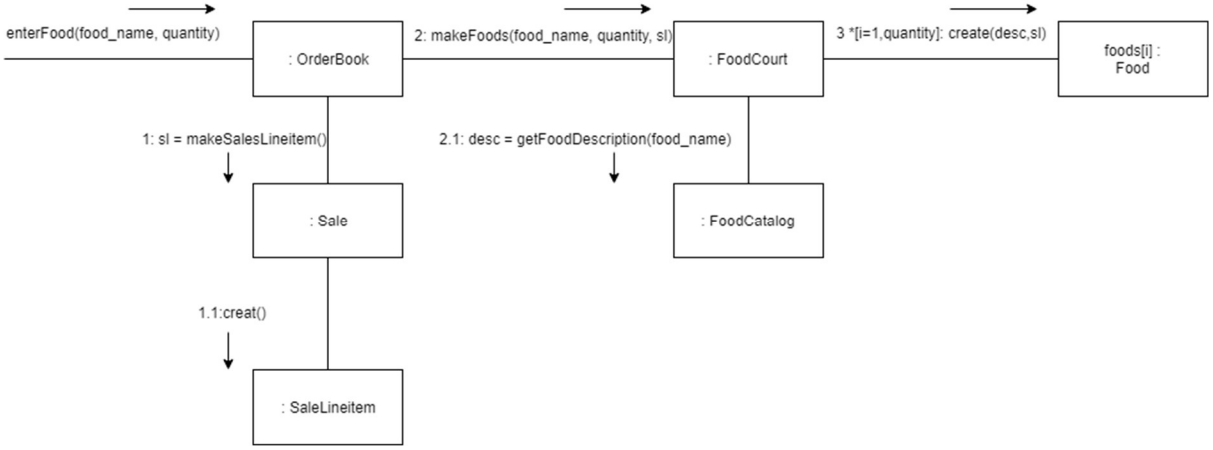
2. Diagram - 주문처리

- **makeNewSale()**



controller패턴에 기반해서 MakeNewSale()메세지를 OrderBook 객체에 보내는것이 적절합니다. 새로운 Sale이 시작되었을 때, creator 패턴에 기반해서 OrderBook이 Sale을 기록하기 때문에 Sale 객체를 생성하는 것이 적절합니다.

- **enterFood(food_name, quantity)**



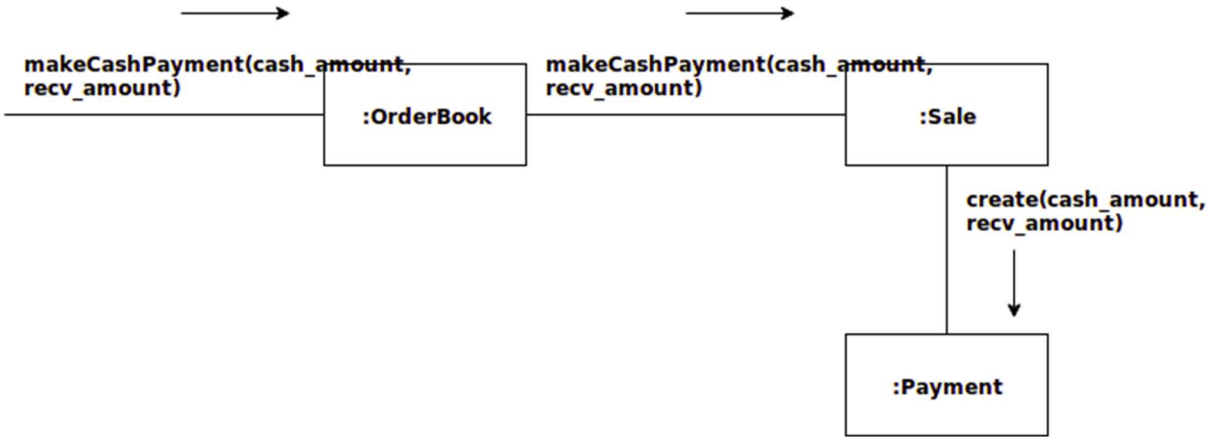
OrderBook class가 주문처리를 대표하는 대표성을 가지기 때문에 public static void Controller pattern에 의하여 주문처리 중 하나인 음식 입력에 해당하는 enterfood()를 실행하는 책임을 주는 것이 적합하다고 생각합니다.

Sale class가 SaleLineitem 객체들을 가장 많이 사용하기 때문에 Creator pattern에 의하여 Sale에게 makeSalesLineitem()을 실행할 책임을 주는 것이 적합하다고 생각합니다.

Foodcourt class가 Food 객체를 생성하는데 필요한 정보를 가장 많이 가지고 있으므로 Information Expert pattern과 Creator pattern에 의하여 FoodCourt에게 makeFoods(food_name, quantity, sl)를 실행할 책임을 주는 것이 적합하다고 생각합니다.

FoodCatalog가 food_name을 attribute로 가지는 FoodDescription객체를 찾는데 필요한 정보를 가장 많이 가지고 있으므로 Information Expert pattern에 의하여 FoodCatalog에게 getFoodDescription(food_name)을 실행할 책임을 주는 것이 적합하다고 생각합니다.

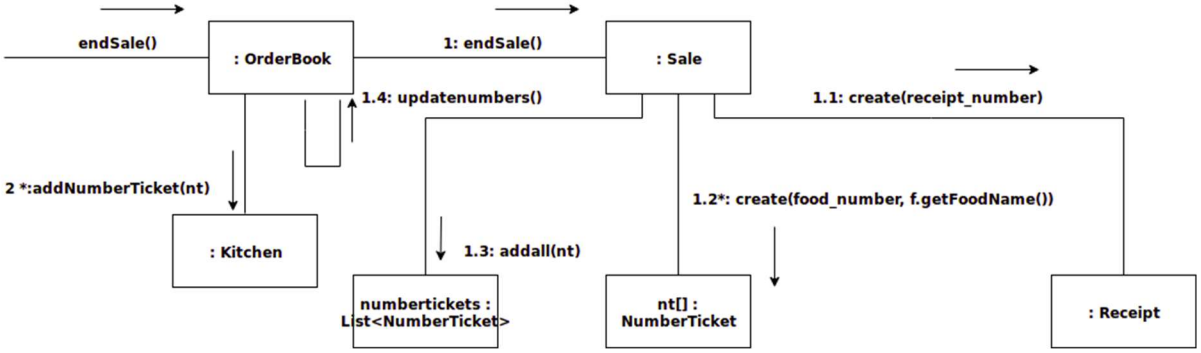
- **makeCashPayment(cash_amount, recv_amount)**



controller패턴에 의해서 OrderBook이 계산을 기록하기 때문에 Orderbook에게 makeCashPayment(cash_amount, recv_amount)를 실행하는 책임을 주는것이 적절합니다.

Sale은 Payment와 밀접하게 사용되고 Sale을 통해 간접적으로 지불내용을 기록할 수 있으므로 Sale에게 makeCashPayment(cash_amount, recv_amount)를 creator 패턴을 이용해서 실행하는 책임을 주는것이 적절합니다. Low Coupling패턴에 의해서 OrderBook과 Sale의 결합도를 낮출 수 있습니다. 이 결과 Payment가 생성되었고 현재 Sale과 association을 갖게 되었습니다.

endSale()

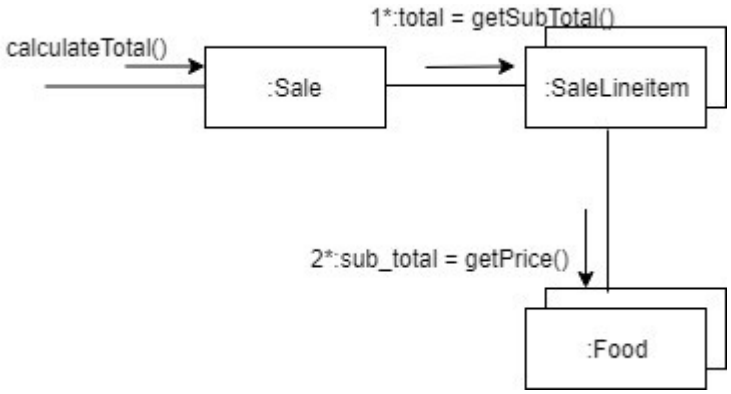


controller패턴에 의해서 판매가 끝났음을 알리는 endSale()의 책임을 OrderBook에 주는것이 적절합니다. 또한 현재의 Sale을 종료시키기위해 Sale클래스에 endSale()의 책임을 주는 것이 적절합니다.

현재의 Sale은 creator패턴에 의해서 Receipt와 Number Ticket의 객체를 생성하는것이 적절합니다. Sale에 의해 새로 생성된 NumberTicket 인스턴스를 List에 추가할수있도록 addall()을 실행할 책임을 information expert 패턴을 이용해서 numertickets에게 주는 것이 적절합니다.

addNumberTicket을 호출하는 책임은 Low Coupling에 의해 OrderBook이 가지는 것이 적절합니다. Sale은 addNumberTicket을 호출하기 위해서 FoodCourt와 Dependency를 맺어야 하지만, OrderBook은 이미 FoodCourt와 Association이 있어서 추가되는 Coupling이 적기 때문입니다.

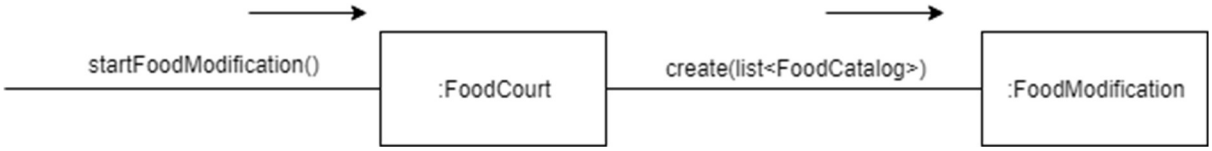
- **calculateTotal**



Information Expert에 의해서 Sale은 계산에 필요한 SaleLineitem을 알고 있으므로, Sale에게 calculateTotal()의 실행 책임을 주는 것이 적절합니다. Imformation Expert에 의해서 price를 Food가 알 고 있으므로, getPrice()의 책임을 Food에게 주는 것이 적절하며, Food 객체를 알 고 있는 SaleLineitem에게 getSubTotal()의 실행 책임을 주는 것이 적절합니다.

3. Diagram - 메뉴관리

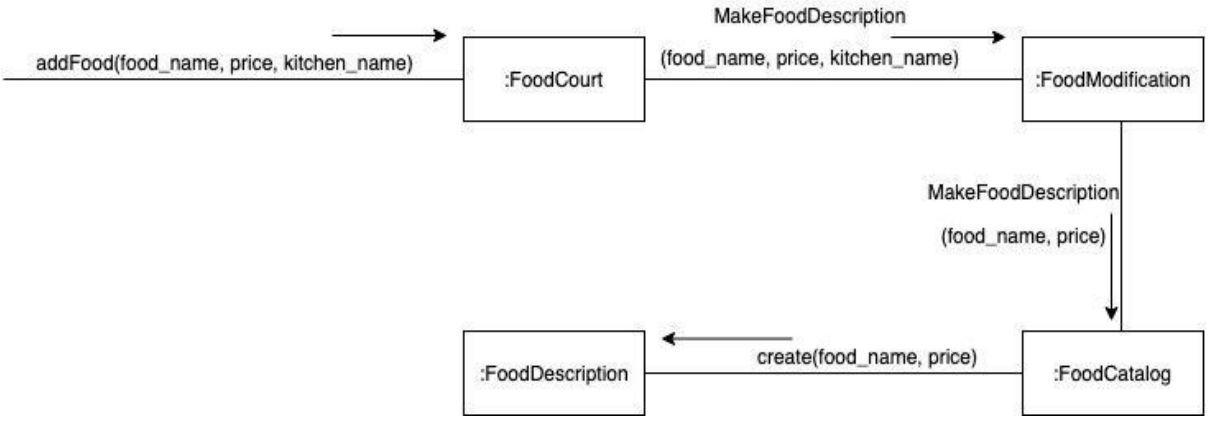
- **startFoodModification()**



FoodCourt class가 하나의 FoodCourt를 대표하는 대표성을 가지기 때문에 Controller pattern에 의하여 FoodCourt의 메뉴관리 중 하나인 메뉴관리 시작에 해당하는 startFoodModification()을 실행하는 책임을 주는 것이 적합하다고 생각합니다.

FoodCourt 객체가 FoodModification 객체를 가장 많이 사용하기 때문에 Creator pattern에 의하여 FoodModification 객체를 생성하는 책임을 FoodCourt 객체에게 주었습니다.

- **addFood(food_name, price, kitchen_name)**



FoodCourt class가 하나의 FoodCourt를 대표하는 대표성을 가지기 때문에 controller 패턴에 의해서 메뉴관리 중 하나인 음식을 추가하는 addFood(food_name, price, kichen_name)를 실행하는 책임을 주는 것이 적절하다고 생각합니다.

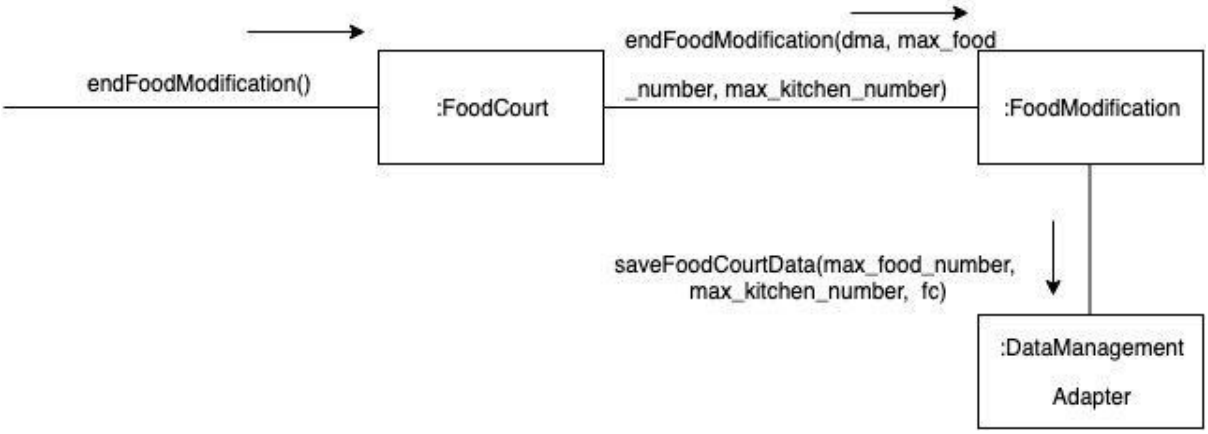
입력된 kitchen_name을 attribute로 가지는 FoodDescription객체가 FoodCatalog에

저장되기 때문에 information expert 패턴에 의해서

MakeFoodDescription(food_name,price)을 실행할 책임을 FoodCatalog에 주는것이 적절하다고 생각합니다.

FoodCatalog가 FoodDescription을 많이 사용하기 때문에 creator패턴에 의해서 FoodDescription에게 create(food_name,price)실행할 책임을 주는것이 적절하다고 생각합니다.

- **endFoodModification()**



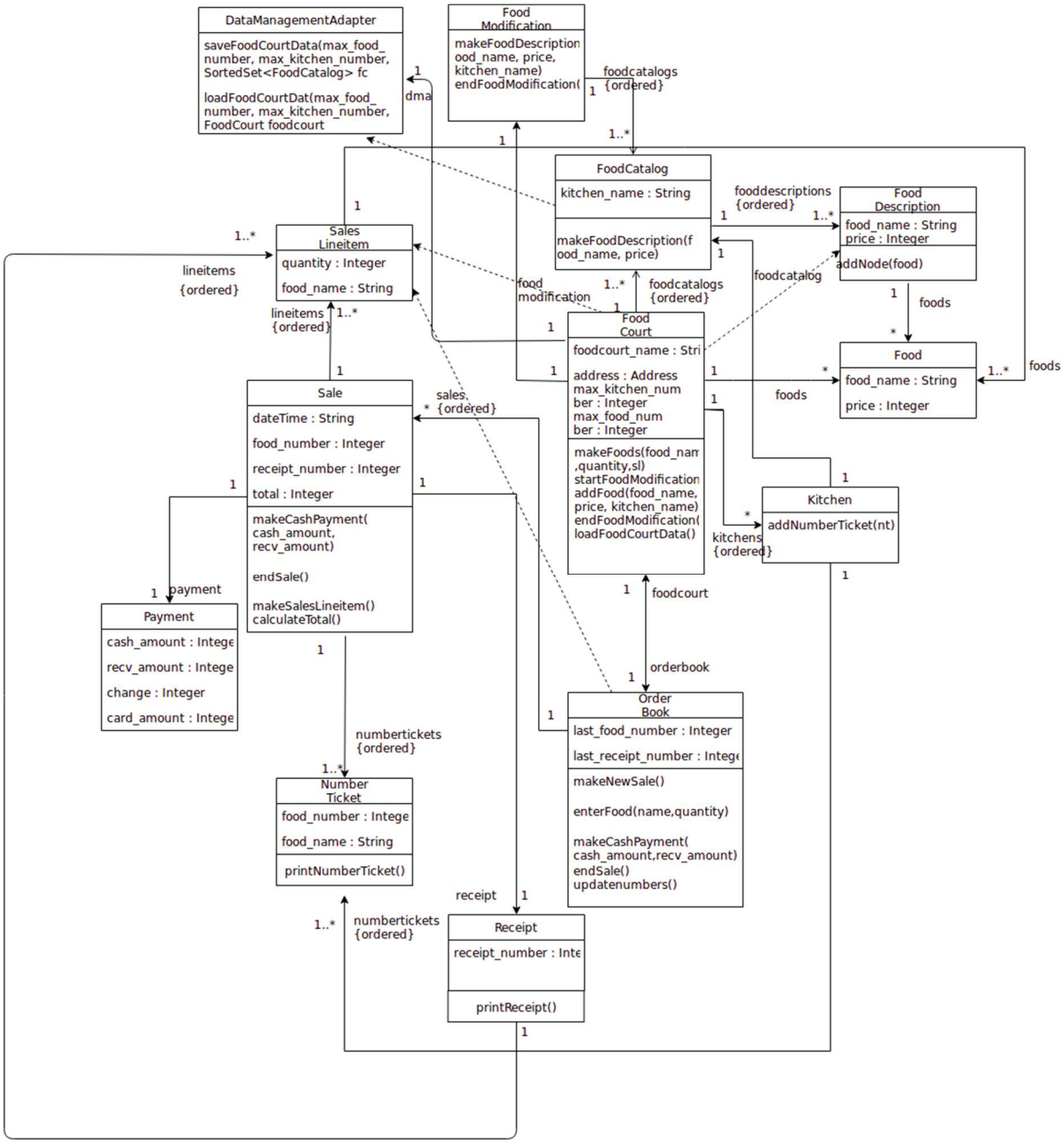
FoodCourt class가 하나의 FoodCourt를 대표하는 대표성을 가지기 때문에 Controller pattern에 의하여 FoodCourt의 메뉴관리 중 하나인 메뉴관리 종료에 해당하는 endFoodModification()을 실행하는 책임을 주는 것이 적합하다고 생각합니다.

FoodCourt 객체가 이미 startFoodModification()에 대한 책임을 가지고 있어서 High Cohesion을 유지하기 위해서 endFoodModification()에 대한 책임도 가져야 한다고 생각합니다.

푸드 코트의 정보를 저장하는 기능은 Iteration이 반복됨에 따라 구현이 추가되는 높은 Evolution을 가집니다. 또한 StakeHolder의 요구에 따라 FileIO를 사용하거나 DataBase를 사용할 수도 있기에 높은 Variation을 가집니다. Protected Variation과 Indirection에 따라 해당 부분을 Pure Fabrication한 Adapter 객체를 거쳐서 실행합니다. 또한, FoodModification이 파일이나 DB 관리의 책임을 Adapter에게 넘겨서 High Cohesion/Low Coupling을 보장할 수 있습니다.

POFS

Design Class Diagram



1. Scope

이번 Design Class가 구현 목표로 삼고 있는 것은, 주문 처리와 주문 관리 Use case의 Main 시나리오입니다.

2. Dependency

FoodCourt과 SalesLineitem의 dependency(parameter visibility)

-> Interaction Diagrams 문서에서 enterFood()를 설명한 interaction diagram에서 FoodCourt 클래스 내에 있는 makeFoods 함수의 매개 변수로 SalesLineitem 객체(sl)가 넘어가기 때문에 parameter visibility입니다.

OrderBook과 SaleLineitem의 dependency(local visibility)

-> Interaction Diagrams 문서에서 enterFood()를 설명한 interaction diagram에서 OrderBook의 method인 enterFood(food_name, quantity) 속에 SaleLineitem을 가리키는 sl 변수를 사용하기 때문에 local visibility입니다.

FoodCourt와 FoodDescription의 dependency(local visibility)

-> Interaction Diagrams 문서에서 enterFood()를 설명한 interaction diagram에서 FoodCourt의 method인 makeFoods(food_name, quantity, sl) 속에 FoodDescription을 가리키는 desc 변수를 사용하기 때문에 local visibility입니다.

DataManagementAdapter와 FoodCatalog의 dependency(parameter visibility)

-> Interaction Diagrams 문서에서 endFoodModification()을 설명한 interaction diagram에서 DataManagementAdapter의 method인 saveFoodCourtData(max_food_number, max_kitchen_number,fc) 속에 FoodCatalog들을 가리키는 Sortedset인 fc 변수를 사용하기 때문에 parameter visibility입니다.

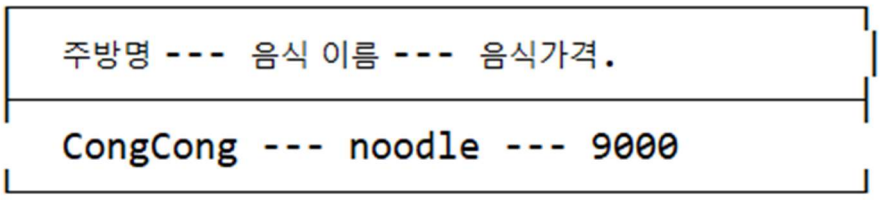
POFS

Test Report

Test case 1(음식추가)

메뉴 관리 use case에서 Manager가 시스템에게 음식 수정을 요청하는 요구 사항을 테스트했습니다. 이 Test case에서는 시스템에 새로운 음식을 추가하는 기능을 보여줍니다.

1. 2 를 입력해서 메뉴 관리를 시작한다.
2. 1 을 입력해서 음식을 추가한다.
3. “noodle”(추가할 음식이름) 을 입력한다.
4. “9000”(추가할 음식가격)을 입력한다.
5. “CongCong”(추가할 음식이 속하는 주방이름)를 입력한다.
6. 주방명, 음식이름, 음식가격이 적힌 음식리스트가 출력된다.
7. 2 를 입력해서 종료한다.

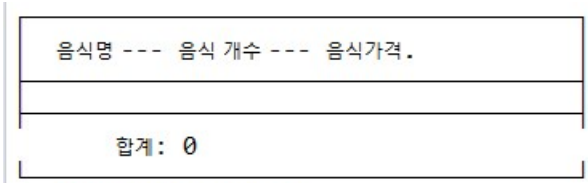


[입력에 성공하였습니다.]
1.음식 추가2.종료

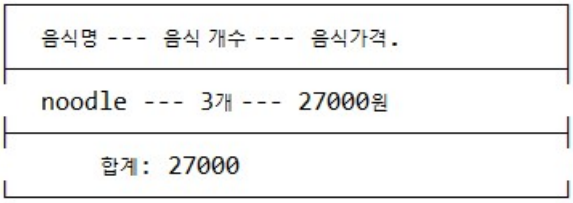
Test case 2 (주문처리)

주문 처리 use case의 main 시나리오에 따라 고객에게 주문을 받고 이를 처리하는 과정이 담긴 Test case입니다. 이 Test case에서는 cashier가 시스템을 통해 고객의 주문을 처리하고 영수증과 번호표를 출력해주는 기능을 보여줍니다.

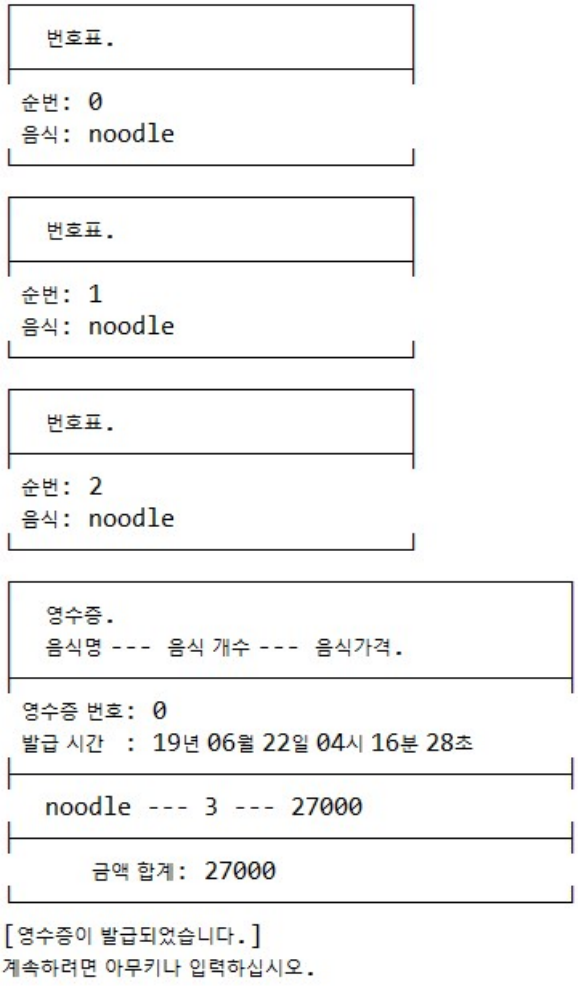
1. 1 을 입력해서 주문처리를 시작한다.
2. 1 을 입력해서 음식을 입력한다.
3. “noodle”(주문할 음식이름)을 입력한다.
4. “3”(주문할 음식의 수량)을 입력한다.
5. 주방명, 음식이름, 음식가격이 적힌 음식리스트가 출력된다.
6. 2 를 입력해서 결제를 한다.
7. 총 금액 “27000”이 출력된다.
8. “27000”(받은 금액)을 입력한다.
9. 거슬러 줄 금액 “0”이 출력된다.
10. 영수증과 번호표가 출력된다.
11. 3 을 입력해서 종료한다.




```
1.음식 입력 2.결제하기 3.종료
1
주문할 음식 이름을 입력하세요.
noodle
주문할 음식의 수량을 입력하세요.
3
```




```
[음식이 입력되었습니다.]
1.음식 입력 2.결제하기 3.종료
2
27000원 결제하시겠습니까?
1.예      2.아니오
1
받은 금액을 입력해주십시오.
27000
거슬러줄 금액은 0 입니다. 계속하려면 아무키나 입력하십시오.
```



Test case 3 (음식추가 후 주문처리)

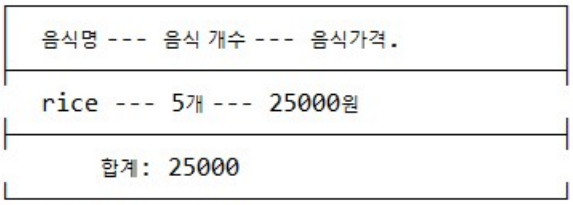
메뉴 관리 use case에서 Manager가 시스템에게 음식 수정을 요청하는 요구 사항을 테스트하고 곧 바로 주문 처리 use case의 main 시나리오에 따라 고객에게 주문을 받는 상황을 테스트했습니다. 이 Test case에서는 시스템에 새로운 음식을 추가하는 기능을 보여주고 시스템을 통해 고객의 주문을 처리하는 기능을 보여줍니다.

1. **2 을 입력해서 메뉴관리를 시작한다.**

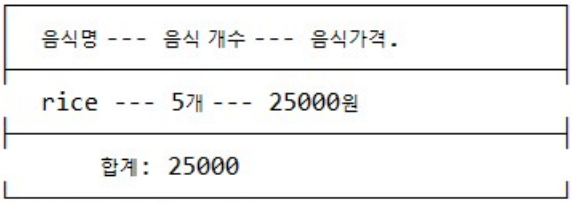
2. 1 을 입력해서 음식을 추가한다.
3. “rice”(추가할 음식이름) 을 입력한다.
4. “5000”(추가할 음식가격)을 입력한다.
5. “CongCong”(추가할 음식이 속하는 주방이름)을 입력한다.
6. 주방명, 음식이름, 음식가격이 적힌 음식리스트가 출력된다.
7. 2 를 입력해서 종료한다.
8. 1 을 입력해서 주문처리를 시작한다.
9. 1 을 입력해서 음식을 입력한다.
10. “rice”(주문할 음식이름)을 입력한다.
11. “5”(주문할 음식의 수량)를 입력한다.
12. 주방명, 음식이름, 음식가격이 적힌 음식리스트가 출력된다.
13. 2 를 입력해서 결제를 한다.
14. 총 금액 “25000”이 출력된다.
15. “30000”(받은 금액)을 입력한다.
16. 거슬러 줄 금액 “5000”이 출력된다.
17. 영수증과 번호표가 출력된다.
18. 3 을 입력해서 종료한다.

```
주방명 --- 음식 이름 --- 음식가격.
노루항 --- 짜장면 --- 6000
노루항 --- 짬뽕 --- 7000
노루항 --- 탕수육 --- 12000
스페셜쉐프 --- 스테이크 --- 9000
스페셜쉐프 --- 스파게티 --- 8000
팔매 --- 된장찌개 --- 8000
팔매 --- 막국수 --- 4000
팔매 --- 순두부찌개 --- 7000
1.음식 추가2.종료
```

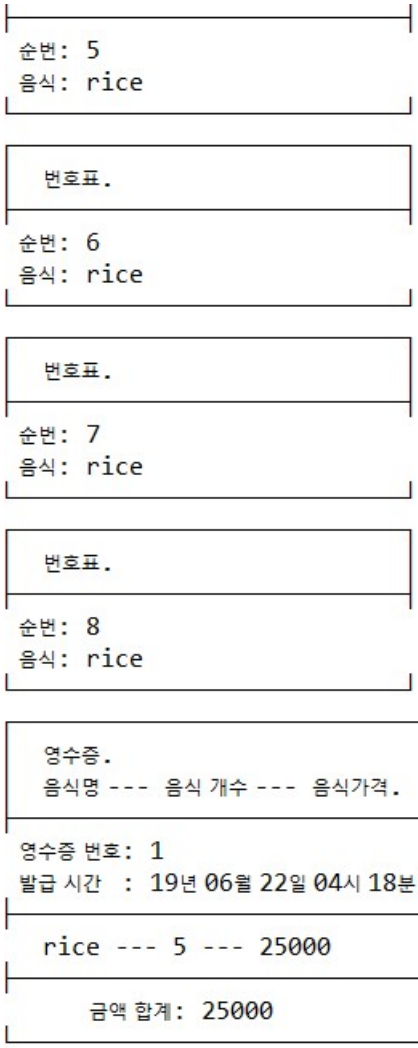
[입력에 성공하였습니다.]
1.음식 추가2.종료



[음식이 입력되었습니다.]
1.음식 입력2.결제하기 3.종료



[음식이 입력되었습니다.]
1.음식 입력2.결제하기 3.종료
2
25000원 결제하시겠습니까?
1.예 2.아니오
1
받은 금액을 입력해주십시오.
30000
거슬러줄 금액은 5000 입니다. 계속하려면 아무키나 입력하십시오.



[영수증이 발급되었습니다.]
계속하려면 아무키나 입력하십시오.

Source code

Address.java

```
package Attribute;

import java.util.Set;



public class Address {

	private String administrative_district;

	private String street_name;

	private Integer building_number;

	private String detailed_address;

}
```

CliFrame.java

```
package CLI;



import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.InputMismatchException;
import java.util.Iterator;
import java.util.Scanner;
import java.util.SortedSet;
import java.util.TreeSet;

import FoodCourt.*;
import Sales.*;
import Output.*;
import Payment.Payment;



//CliFrame의 함수들은 사용자가 보고 입력하는 부분만 담당합니다.
public class CliFrame {
	private static ArrayList<String> cli_list_first_col = new ArrayList<String>();
```

```
private static ArrayList<String> cli_list_second_col = new ArrayList<String>();
private static ArrayList<String> cli_list_third_col = new ArrayList<String>();

/*-------------------------------------------------------
 *
 * 공통메뉴 부분.
 *
 *-------------------------------------------------------*/
public static void clearConsole()
{
	for(int i =0; i < 300; i++) System.out.println("");
}


public static void showStartMenuUi(FoodCourt foodcourt)
{
	clearConsole();
	Scanner sc = new Scanner(System.in);
	boolean isExit = false;
	char input = 'a'; String str;


	while(isExit == false)
```

```
        {

System.out.println("┌──────────────────────────────────────────────────────────────────────────────┐
");
                System.out.println("|                                                              |");
                System.out.println("|                POFS                |");
                System.out.println("|                                                              |");

System.out.println("└──────────────────────────────────────────────────────────────────────────────┘
");
                System.out.println("선택하실 메뉴 번호를 입력해주세요.");
                System.out.println("1.주문처리₩t2.메뉴관리₩t4.종료");
                str = sc.nextLine();
                if(!str.equals(""))input = str.charAt(0);
                clearConsole();

                switch(input)
                {
                case '1':
                        showOrderProcessUi(foodcourt.getOrderbook());
                        break;
                case '2':
                        showMenuManagementUi(foodcourt);
                        break;
```

```
                case '3':
                        //다음에 구현될 부분.
                        break;
                case '4':
                        isExit = true;
                        sc.close();
                        break;
                default:

                        System.out.println("[올바르지 않은 입력입니다.]");
                        break;
                }
        }
}



/*-------------------------------------------------------------
 *
 *
 * 주문 처리 파트.
 *
 *
 * -------------------------------------------------------------*/
```

```
public static void showOrderProcessUi(OrderBook orderbook)
{
	Scanner sc = new Scanner(System.in);
	boolean is_exit = false;
	int food_number = 0, food_price = 0;
	String food_name = ""; int input_price;
	char input = 'a'; boolean is_food_exists = true;
	int total = 0;
	String str;
	clearConsole();




	showOrderedFoodList(total);
	orderbook.makeNewSale();
	Sale cur_sale = orderbook.getSales().last();

	while(is_exit == false)
	{

		System.out.println("1.음식  입력₩t2.결제하기₩t3.종료");
		str = sc.nextLine();
```

```
if(!str.equals(""))input = str.charAt(0);

switch(input)
{
case '1':
	try
	{
		//고객이 Cashier 에게 주문할 음식을 말한다.
		System.out.println("주문할 음식 이름을 입력하세요.");
		food_name = sc.nextLine();
		System.out.println("주문할 음식의 수량을 입력하세요.");
		food_number = sc.nextInt(); sc.nextLine();

		if(is_food_exists && (food_number < 100) && (food_number
> 0))
		{

			FoodDescription fd =
orderbook.getFoodcourt().getFoodCatalogByFoodName(food_name).getFoodDescription(food_nam
e);
			food_price = fd.getPrice();


			orderbook.enterFood(food_name, food_number);
```

```
                                addOrderedFoodList(food_name, food_number,
food_price*food_number);
                                //CLI 리스트에 음식을 추가합니다. 단순히 보여주는
리스트에 추가하는 역할입니다.
                                clearConsole();

                                cur_sale.calculateTotal();
                                total = cur_sale.getTotal();
                                showOrderedFoodList(total);
                                System.out.println("[음식이 입력되었습니다.]");
                                //주문한 음식을 처리하는 루틴이 들어가야 합니다.
isFoodExists를 통해 해당 음식이 존재하는 음식인지
                                //알려줘야합니다. food_price에 음식의 가격을 알려줘
야 합니다.

                        }
                        else
                        {
                                clearConsole();
                                showOrderedFoodList(total);
                                System.out.println("[잘못된 입력입니다.]");
                        }
                }
                catch(InputMismatchException e)
```

```
                {
                        sc.nextLine();
                        clearConsole();
                        showOrderedFoodList(total);
                        System.out.println("[수량은 정수로 입력되어야 합니다.]");
                }
                break;
        case '2':
                try
                {
                        //Cashier가 고객에게 총 가격을 말해주고 결제를 요청한다.

                        cur_sale.calculateTotal();
                        total = cur_sale.getTotal();
                        System.out.printf("%d원 결제하시겠습니까?₩n1.예₩t2.아니오
₩n", total);
                        input = sc.nextLine().charAt(0);

                        if(input != '1')
                        {
                                clearConsole();
                                showOrderedFoodList(total);
                                System.out.println("[결제를 취소하였습니다.]");
```

```
                    break;
                }
                //Cashier가 받은 금액을 입력한다.
                System.out.println("받은 금액을 입력해주십시오.");
                input_price = sc.nextInt(); sc.nextLine();

                orderbook.makeCashPayment(total, input_price);
                Payment cur_payment = cur_sale.getPayment();
                //시스템이 거슬러줄 금액을 보여준다.
                System.out.printf("거슬러줄 금액은 %d 입니다. 계속하려면 아무키나 입력하십시오.₩n", cur_payment.getChange());
                sc.nextLine();

                //시스템이 완료된 판매를 기록하고 매출에 반영한 뒤 주방으로 전달해준다.

                orderbook.endSale();

                //시스템이 영수증을 출력한다.
                System.out.printf("계속하려면 아무키나 입력하십시오.₩n");
```

```
                sc.nextLine();



                //시스템이 주방과 계산대에 번호표를 출력한다.
                //계산대의 영수증과 번호표는 빠른 처리를 위해서 CLI가 들고있는 리스트를 변수로 받고 있습니다.
                //Domain Layer 설계시 따로 건드릴 필요가 없습니다.

                //Cashier가 판매 완료 처리를 한다.

                clearConsole();

                cli_list_first_col = new ArrayList<String>();
                cli_list_second_col = new ArrayList<String>();
                cli_list_third_col = new ArrayList<String>();

                total = 0;
                showOrderedFoodList(total);
                System.out.println("[결제를 완료하였습니다.]");
                orderbook.makeNewSale();
                cur_sale = orderbook.getSales().last();

            }
```

```
                catch(InputMismatchException e)
                {
                        clearConsole();
                        showOrderedFoodList(total);
                        System.out.println("[가격은 정수로 입력되어야 합니다.]");
                }
                break;
        case '3':
                clearConsole();

                cli_list_first_col = new ArrayList<String>();
                cli_list_second_col = new ArrayList<String>();
                cli_list_third_col = new ArrayList<String>();

                is_exit = true;
                break;
        default:
                clearConsole();
                showOrderedFoodList(total);
                System.out.println("[잘못된 입력입니다.]");
        }

}
```

```
    }

    private static void addOrderedFoodList(String food_name, int food_number, int food_price)
    {
        cli_list_first_col.add(food_name);
        cli_list_second_col.add(String.format("%d개", food_number));
        cli_list_third_col.add(String.format("%d원", food_price));
    }

    public static void showOrderedFoodList(int total)
    {

System.out.println(" ┌──────────────────────────────────────────────────────────────────────
──────┐ ");
        System.out.println(" |    음식명 --- 음식 개수 --- 음식가격.              | ");

System.out.println(" ├──────────────────────────────────────────────────────────────────────
──────┤ ");
        for(int i =0; i < cli_list_first_col.size(); i++)
        {
            System.out.printf("     %s --- %s --- %s₩n", cli_list_first_col.get(i),
                                cli_list_second_col.get(i), cli_list_third_col.get(i));
        }
```

```
System.out.println(" ├──────────────────────────────────────
────┤ ");
            System.out.printf("                    합계: %d₩n", total);

System.out.println(" └──────────────────────────────────────
────┘ ");
    }

    public static void printRecepit(int receipt_number, int total, SortedSet<SaleLineitem> lineitems)
    {
            String time;
            SimpleDateFormat time_format = new SimpleDateFormat("yy년 MM월 dd일 hh시 mm
분 ss초");
            Calendar calender = Calendar.getInstance();
            time = time_format.format(calender.getTime());
            SaleLineitem sli;


System.out.println(" ┌──────────────────────────────────────
────┐ ");
            System.out.println(" │   영수증.                                   │ ");
            System.out.println(" │   음식명 --- 음식 개수 --- 음식가격.          │ ");

System.out.println(" ├──────────────────────────────────────
────┤ ");
```

```
            System.out.printf("     영수증 번호: %d₩n", receipt_number);
            System.out.printf("     발급 시간    : %s₩n", time);

System.out.println(" ├──────────────────────────────────────────────────────────────────────
──────┤ ");

            Iterator<SaleLineitem> it = lineitems.iterator();
            while(it.hasNext()) {
                  sli = it.next();
                  System.out.printf("     %s --- %s --- %s₩n", sli.getfoodname(),
                                    sli.getquantity(), sli.getSubTotal());
            }

System.out.println(" ├──────────────────────────────────────────────────────────────────────
──────┤ ");
            System.out.printf("                    금액 합계: %d₩n", total);

System.out.println(" └──────────────────────────────────────────────────────────────────────
──────┘ ");
            System.out.println("[영수증이 발급되었습니다.]");
      }

      public static void printNumberTicket(int food_number, String food_name)
      {
```

```
System.out.println(" ┌──────────────────────────────────────────────────────────┐ ");
                    System.out.println(" |     번호표.                              | ");
System.out.println(" ├──────────────────────────────────────────────────────────┤ ");
                    System.out.printf("     순번: %d₩n", food_number);
                    System.out.printf("     음식: %s₩n", food_name);
System.out.println(" └──────────────────────────────────────────────────────────┘ ");
    }


    /*-------------------------------------------------------
     *
     * 메뉴 관리 파트
     *
     *
          ------------------------------------------------------------*/

    public static void showMenuManagementUi(FoodCourt foodcourt)
    {
          char input = 'a';       boolean is_exit = false;
          Scanner sc = new Scanner(System.in);
          String food_name, kitchen_name; int food_price;
```

```
boolean is_exists_kitchen_name = true;
String str;



initKitchenNFoodList(foodcourt);
clearConsole();
showKitchenNFoodList();



foodcourt.startFoodModification();



while(is_exit == false)
{
	System.out.println("1.음식 추가₩t2.종료");
	str = sc.nextLine();
	if(!str.equals(""))input = str.charAt(0);



	switch(input)
	{
	case '1':
		try
```

```
                    {
                        //Manager가 추가할 음식의 이름과 가격을 입력한다.
Manager가 추가할 음식이 속하는 주방 이름을 입력한다.
                        System.out.println("추가할 음식의 이름을 입력하세요.");
                        food_name = sc.nextLine();
                        System.out.println("추가할 음식의 가격을 입력하세요.");
                        food_price = sc.nextInt(); sc.nextLine();
                        System.out.println("추가할 음식이 속하는 주방을 입력하세
요.");
                        kitchen_name = sc.nextLine();
                        //시스템이 입력된 정보를 통해서 주방과 음식의 리스트를 업
데이트한다.

                        if(is_exists_kitchen_name)
                        {
                            addKitchenNFoodList(kitchen_name, food_name,
food_price);
                            foodcourt.addFood(food_name, food_price,
kitchen_name);
                            //실제 입력된 정보를 DCD에 있는 적절한 객체에 추
가하는 루틴이 필요합니다.

                            //시스템이 Manager에게 업데이트된 주방과 음식의
리스트를 보여준다.
                            clearConsole();
```

```
                        showKitchenNFoodList();
                        System.out.println("[입력에 성공하였습니다.]");
                }
                else
                {
                        //아직 고려하지 않는 부분입니다.(익스텐션 시나리오)
                }
        }
        catch(InputMismatchException e)
        {
                sc.nextLine();
                clearConsole();
                showKitchenNFoodList();
                System.out.println("[가격은 정수로 입력되어야 합니다.]");
        }
        break;
case '2':
        //실제 입력된 정보를 저장하는 루틴이 필요합니다.

        is_exit = true;
        cli_list_first_col = new ArrayList<String>();
        cli_list_second_col = new ArrayList<String>();
        cli_list_third_col = new ArrayList<String>();
```

```
                        foodcourt.endFoodModification();

                        clearConsole();

                        break;

                default:

                        clearConsole();

                        showKitchenNFoodList();

                        System.out.println("[잘못 입력되었습니다.]");

                        break;

                }

        }

    }


    public static void showKitchenNFoodList()

    {

System.out.println("┌──────────────────────────────────────────────────────────
────┐");

        System.out.println("|    주방명 --- 음식 이름 --- 음식가격.            | ");

System.out.println("├──────────────────────────────────────────────────────────
────┤");

        for(int i =0; i < cli_list_first_col.size(); i++)

        {

                System.out.printf("    %s --- %s --- %s₩n", cli_list_first_col.get(i),
```

```
                                        cli_list_second_col.get(i), cli_list_third_col.get(i));
            }

System.out.println(" └──────────────────────────────────────────────────────────────────────────────
──────┘ ");
    }

    private static void addKitchenNFoodList(String kitchen_name, String food_name, int food_price)
    {
            cli_list_first_col.add(kitchen_name);
            cli_list_second_col.add(food_name);
            cli_list_third_col.add(String.format("%s", food_price));
    }

    public static void initKitchenNFoodList(FoodCourt foodcourt)
    {
            SortedSet<FoodCatalog> fc = foodcourt.getAllFoodCatalog();
            String food_name, kitchen_name; int food_price;

            Iterator<FoodCatalog> cit = fc.iterator();
            Iterator<FoodDescription> dit;
            FoodCatalog cur = null; FoodDescription fd = null;
            int max_food_number = foodcourt.getMaxFoodNumber();
```

```
        int max_kitchen_number = foodcourt.getMaxKitchenNumber();
        int index = 0;
        int inner_index = 0;

        SortedSet<FoodDescription> fooddescriptions = new TreeSet<FoodDescription>();

        while(cit.hasNext() && index < max_kitchen_number) {
                cur = cit.next();
                kitchen_name = cur.getKitchen_name();
                fooddescriptions = cur.getAllFooddescriptions();
                dit = fooddescriptions.iterator();
                while(dit.hasNext() && inner_index < max_food_number) {
                        fd = dit.next();
                        food_name = fd.getFood_name();
                        food_price = fd.getPrice();
                        addKitchenNFoodList(kitchen_name, food_name, food_price);
                        inner_index++;
                }

                index++;
        }
}
```

```
}
```

Food.java

```
package FoodCourt;

public class Food implements Comparable<Food> {

	private String food_name;
	private Integer price;

	//접근 함수들입니다.
	public String getFoodName() {
		return food_name;
	}

	public Integer getPrice() {
		return price;
	}

	//접근 함수들을 제외한 함수들입니다.

	public Food(String food_name, Integer price) {

		this.food_name = food_name;
```

```
        this.price = price;
    }

    @Override
    public int compareTo(Food o) {
        return this.food_name.compareTo(o.getFoodName());
    }

}
```

FoodCatalog.java

```
package FoodCourt;

import java.util.Iterator;

import java.util.SortedSet;

import java.util.TreeSet;


public class FoodCatalog implements Comparable<FoodCatalog> {



	private String kitchen_name = "";

	private SortedSet<FoodDescription> fooddescriptions;


	//접근함수들입니다.

	public FoodCatalog(String kitchen_name) {

		this.kitchen_name = kitchen_name;

		fooddescriptions = new TreeSet<FoodDescription>();

	}



	public String getKitchen_name() {


		return this.kitchen_name;
```

```
}

public SortedSet<FoodDescription> getAllFooddescriptions() {
	return fooddescriptions;
}

public void setFooddescriptions(SortedSet<FoodDescription> fooddescriptions) {
	this.fooddescriptions = fooddescriptions;
}

public FoodDescription getFoodDescription(String food_name) {

	Iterator<FoodDescription> seek = fooddescriptions.iterator();
	FoodDescription A = null;

	while(seek.hasNext()) {
		A =seek.next();
		if(A.getFood_name().equals(food_name))
			return A;

	}
	return null;
}
```

```
    //접근 함수를 제외한 함수들입니다.
    public void makeFoodDescription(String food_name, Integer price) {
        FoodDescription A = new FoodDescription(food_name, price);
        this.fooddescriptions.add(A);
    }

    @Override
    public int compareTo(FoodCatalog o) {
        return this.kitchen_name.compareTo(o.getKitchen_name());
    }
}
```

FoodCourt.java

```
package FoodCourt;

import java.util.SortedSet;

import java.util.TreeSet;


import Attribute.Address;

import CLI.CliFrame;


import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.Set;

import Sales.*;

import TechnicalSupport.*;


public class FoodCourt {


    private static final int max_kitchen_number = 15;

    private final static int max_food_number = 150;

    private String foodcourt_name;

    private Address address;

    private FoodModification foodmodification;
```

```
private SortedSet<FoodCatalog> foodcatalogs;
private List<Food> foods;
private OrderBook orderbook;
private Set<Kitchen> kitchens;
private DataManagementAdapter dma;

//접근함수들입니다.
public int getMaxKitchenNumber() {
	return max_kitchen_number;
}

public int getMaxFoodNumber() {
	return max_food_number;
}

public Set<Kitchen> getKitchenSet() {
	return kitchens;
}

public Kitchen getKichen(FoodCatalog fc) {
	Iterator<Kitchen> seek = this.kitchens.iterator();
	Kitchen A = null;
```

```
        while(seek.hasNext()) {
                A =seek.next();
                if(A.getFoodCatalog().equals(fc))
                        return A;
        }
        return null;
}

public SortedSet<FoodCatalog> getAllFoodCatalog() {
        return this.foodcatalogs;
}

public FoodCatalog getFoodCatalog(String kitchen_name) {

        Iterator<FoodCatalog> seek = this.foodcatalogs.iterator();
        FoodCatalog A = null;

        while(seek.hasNext()) {
                A =seek.next();
                if(A.getKitchen_name().equals(kitchen_name))
                        return A;
        }
        return null;
```

```
}

public FoodCatalog getFoodCatalogByFoodName(String food_name) {

	FoodCatalog fc = null;

	Iterator <FoodCatalog> it = foodcatalogs.iterator();
	FoodCatalog catalogcur = null;

	while(it.hasNext()) {
		catalogcur = it.next();

		if(catalogcur.getFoodDescription(food_name) != null) {
			return catalogcur;
		}
	}

	return null;
}
//입력받은 음식 이름을 가지는 포함하는 푸드 카탈로그를 반환해야합니다.
```

```
public OrderBook getOrderbook() {
	return orderbook;
}


//접근함수를 제외한 함수들입니다.
public static void main(String[] args) {
	FoodCourt foodcourt = new FoodCourt(null, null);
	foodcourt.loadFoodCourtData();
	CliFrame.showStartMenuUi(foodcourt);
}



public FoodCourt(String foodcourt_name, Address address) {

	this.foodcourt_name = foodcourt_name;
	this.address = address;
	this.orderbook = new OrderBook(this);
	FoodCatalog fc;
	Integer itmp;
	this.foodcatalogs = new TreeSet<FoodCatalog>();
```

```
        this.foods = new ArrayList<>();
        this.kitchens = new TreeSet<Kitchen>();

}

public void loadFoodCourtData() {
        if(dma == null) dma = new DataManagementAdapter();
        this.dma.loadFoodCourtData(max_food_number, max_kitchen_number, this);
}

public void makeFoods(String food_name, Integer quantity, SaleLineitem sl) {

        Iterator<FoodCatalog> seek = foodcatalogs.iterator();
        FoodDescription desc = null;
        FoodCatalog temp = null;

        while(seek.hasNext()) {
                temp = seek.next();
                desc = temp.getFoodDescription(food_name);
```

```
                if(desc.equals(null))
                        continue;

                else {
                        for(int i =1; i<=quantity; i++) {
                                Food food = new Food(food_name,desc.getPrice()); //
                                this.foods.add(food);
                                desc.addnode(food);
                                sl.addnode(food);
                        }
                        break;
                }
        }
}

public void startFoodModification() {
        foodmodification = new FoodModification(foodcatalogs);
}

public void addFood(String food_name, Integer price, String kitchen_name) {
        foodmodification.makeFoodDescription(food_name, price, kitchen_name);
}
```

```
	public void endFoodModification() {

		foodmodification.endFoodModification(this.dma, this.max_food_number,
this.max_kitchen_number);

	}

}
```

FoodDescription.java

```
package FoodCourt;

import java.util.ArrayList;

import java.util.List;

import java.util.SortedSet;

import java.util.TreeSet;


public class FoodDescription implements Comparable<FoodDescription>{



	private String food_name;

	private Integer price;

	private List<Food> foods;


	public FoodDescription(String food_name, Integer price) {

		this.food_name = food_name;

		this.price = price;

		this.foods = new ArrayList<Food>();


	}


	public Integer getPrice() {
```

```
        return price;
    }


    public String getFood_name() {
        return food_name;
    }

    public void addnode(Food food) {

        this.foods.add(food);
    }


    @Override
    public int compareTo(FoodDescription o) {
        return this.food_name.compareTo(o.getFood_name());
    }
}
```

FoodModification.java

```
package FoodCourt;
import java.util.Iterator;
import java.util.SortedSet;
import java.util.TreeSet;
import TechnicalSupport.*;

public class FoodModification {
	private SortedSet<FoodCatalog> foodcatalogs;

	public FoodModification(SortedSet<FoodCatalog> foodcatalogs) {

		this.foodcatalogs = foodcatalogs;
	}
	private FoodCatalog searchFoodCatalog(String kitchen_name) {
		Iterator<FoodCatalog> it = foodcatalogs.iterator();
		FoodCatalog cur = null;

		while(it.hasNext()) {
			cur = it.next();
			if(cur.getKitchen_name().equals(kitchen_name)) return cur;
		}
```

```
            return null;
        }


        public void endFoodModification(DataManagementAdapter dma, int max_food_number, int
max_kitchen_number) {
            dma.saveFoodCourtData(max_food_number, max_kitchen_number, this.foodcatalogs);
        }

        public void makeFoodDescription(String food_name, Integer price, String kitchen_name) {
            FoodCatalog fc = searchFoodCatalog(kitchen_name);

            if(fc != null) {
                fc.makeFoodDescription(food_name, price);
            } else {

            }
        }
}
```

Kitchen.java

```
package FoodCourt;

import java.util.SortedSet;
import java.util.TreeSet;

import Output.NumberTicket;
import Sales.Sale;



public class Kitchen implements Comparable<Kitchen> {
	private FoodCatalog foodcatalog;
	private SortedSet<NumberTicket> numbertickets;

	public FoodCatalog getFoodCatalog() {
		return foodcatalog;
	}

	public void addNumberTicket(NumberTicket nt) {
		numbertickets.add(nt);
	}
```

```
    public Kitchen(FoodCatalog fc) {
        this.foodcatalog = fc;
        this.numbertickets = new TreeSet<NumberTicket>();
    }

    public int compareTo(Kitchen k) {
        return this.foodcatalog.getKitchen_name().compareTo(k.foodcatalog.getKitchen_name());
    }
}
```

NumberTicket.java

```
package Output;

import java.util.SortedSet;

import CLI.CliFrame;
import FoodCourt.FoodDescription;

public class NumberTicket implements Comparable<NumberTicket>{
	private Integer food_number;
	private String food_name;



		public Integer getFood_number() {
		 return food_number;
	}



	public String getFood_name() {
		 return food_name;
```

```
    }



    public NumberTicket(Integer food_number, String food_name) {
        this.food_number = food_number;
        this.food_name = food_name;
    }

    public void printNumberTicket() {
         CliFrame.printNumberTicket(this.food_number, this.food_name);
    }

    @Override
    public int compareTo(NumberTicket o) {
         return this.food_number.compareTo(o.getFood_number());
    }


}
```

Receipt.java

```
package Output;

import java.util.List;
import java.util.SortedSet;

import CLI.CliFrame;
import Sales.SaleLineitem;

public class Receipt {

	private Integer receipt_number;
	private Integer total;
	private SortedSet<SaleLineitem> lineitems;

	 public Integer getReceiptNumber() {
		return receipt_number;
	 }

	public Receipt(Integer receipt_number, int total, SortedSet<SaleLineitem> lineitem) {
		this.receipt_number = receipt_number;
		this.total = total;
```

```
            this.lineitems = lineitem;
        }


        public void printReceipt() {
                CliFrame.printRecepit(this.receipt_number, this.total, this.lineitems);
        }



}
```

Payment.java

```
private Integer change;
        private Integer card_amount;


    public Integer getCashAmount()
    {
        return cash_amount;
    }


    public Integer getRecvAmount()
    {
        return recv_amount;
    }


    public Integer getChange()
    {
            return change;
    }
```

```
  public Payment(Integer cash_amount, Integer recv_amount)
 {
     this.recv_amount = recv_amount;
     this.cash_amount = cash_amount;
     this.change = recv_amount - cash_amount;
 }


 public Integer getCashAmount()
 {
     return cash_amount;
}

 public Integer getRecvAmount()
{
     return recv_amount;
 }


 public Integer getChange()
 {
      return change;
```

```
        }

    }
```

OrderBook.java

```
package Sales;

import java.util.Iterator;

import java.util.SortedSet;

import java.util.TreeSet;


import FoodCourt.Food;

import FoodCourt.FoodCourt;

import Output.NumberTicket;



public class OrderBook {

	private Integer last_food_number;

	private Integer last_receipt_number;

	private SortedSet<Sale> sales;

	private FoodCourt foodcourt;



	public OrderBook(FoodCourt foodcourt) {

		sales = new TreeSet<>();

		last_food_number = 0;

		last_receipt_number =0;

		this.foodcourt = foodcourt;
```

```
}

public SortedSet<Sale> getSales() {
        return sales;
}

public FoodCourt getFoodcourt() {
        return foodcourt;
}

public void makeNewSale() {
        sales.add(new Sale(last_food_number, last_receipt_number));
}


public void enterFood(String food_name, Integer quantity) {
        SaleLineitem sl = sales.last().makeSaleLineitem();
        foodcourt.makeFoods(food_name, quantity, sl);
}


public void makeCashPayment(Integer cash_amount, Integer recv_amount) {
        sales.last().makeCashPayment(cash_amount,recv_amount);
```

```
	}

	public void endSale() {
		sales.last().endSale();

		if(sales.size() != 0) {
			updatenumbers();
		}

		Iterator<NumberTicket> it = sales.last().getNumbertickets().iterator();
		NumberTicket nt;

		while(it.hasNext()) {
			nt = it.next();

foodcourt.getKichen((foodcourt.getFoodCatalogByFoodName(nt.getFood_name()))).addNumberTick
et(nt);

			}
	}
```

```
    public void updatenumbers() {
        last_receipt_number = sales.last().getreceiptnumber() + 1;
        last_food_number = sales.last().getfoodnumber() + 1;
    }
}
```

Sale.java

```
package Sales;

import java.util.SortedSet;

import java.util.TreeSet;


import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Date;

import java.util.Iterator;

import java.util.List;


import Output.*;

import Payment.*;

import FoodCourt.*;


public class Sale implements Comparable<Sale>{


    private SortedSet<NumberTicket> numbertickets;

    private Receipt receipt;

    private Payment payment;

    private SortedSet<SaleLineitem> lineitems;
```

```
private String dateTime;
private Integer food_number;
private Integer receipt_number;
private Integer total;

//접근함수들입니다.

public Payment getPayment() {
	return this.payment;
}

public Integer getTotal() {
	return this.total;
}

public SortedSet<NumberTicket> getNumbertickets() {
	return numbertickets;
}

public Integer getfoodnumber() {
	return food_number;
}
```

```
public Integer getreceiptnumber() {
        return receipt_number;
}

//접근을 제외한 함수들입니다.

public Sale(int last_food_number, int last_receipt_number) {
        this.total = 0;
        Date from = new Date();
        SimpleDateFormat transFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
        this.dateTime = transFormat.format(from);
        this.food_number = last_food_number;
        this.receipt_number = last_receipt_number;
        this.lineitems = new TreeSet<>();
        this.numbertickets = new TreeSet<>();
}

public void makeCashPayment(Integer cash_amount, Integer recv_amount) {
        payment = new Payment(cash_amount, recv_amount);
}

public void endSale() {
```

```
NumberTicket ntcur;

Iterator <SaleLineitem> it = lineitems.iterator();
SaleLineitem cur = null;

SortedSet<NumberTicket> nt = new TreeSet<>();
Food f;

while(it.hasNext()) {
        cur = it.next();

        Iterator <Food> itt = cur.getfoods().iterator();
        while(itt.hasNext()) {
                f = itt.next();
                ntcur = new NumberTicket(food_number, f.getFoodName());
                this.food_number += 1;
                nt.add(ntcur);
                ntcur.printNumberTicket();
        }
}

numbertickets.addAll(nt);
```

```
        calculateTotal();
        this.receipt = new Receipt(receipt_number, this.total, this.lineitems);
        this.receipt.printReceipt();
}

public SaleLineitem makeSaleLineitem() {
        SaleLineitem temp = new SaleLineitem();
        this.lineitems.add(temp);
        return temp;
}


public int compareTo(Sale sale) {
        return this.dateTime.compareTo(sale.dateTime);
}

public void calculateTotal()
{
        int total = 0;
        Iterator<SaleLineitem> it = this.lineitems.iterator();
        SaleLineitem sli;
        while(it.hasNext()) {
                sli = it.next();
```

```
                total += sli.getSubTotal();
            }

            this.total = total;
        }

}
```

SaleLineitem.java

```
package Sales;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.SortedSet;
import java.util.TreeSet;
import FoodCourt.*;
import Output.NumberTicket;

public class SaleLineitem implements Comparable<SaleLineitem>{


	private Integer quantity;
	private List<Food> foods;
	private String food_name;

	//접근함수입니다.
	public Integer getquantity() {
		return quantity;
	}
```

```
public String getfoodname() {
        return food_name;
}

public List<Food> getfoods() {
        return foods;
}

//접근함수를 제외한 함수들입니다.
public SaleLineitem() {
        quantity = 0;
        this.foods = new ArrayList<Food>();
}



public void addnode(Food food) {
        this.food_name = food.getFoodName();
        this.quantity += 1;
        foods.add(food);
}
```

```
	public int getSubTotal() {

		Iterator <Food> it = this.foods.iterator();
		Food f; int sub_total = 0;

		while(it.hasNext()) {
			f = it.next();
			sub_total += f.getPrice();
		}

		return sub_total;
	}

	@Override
	public int compareTo(SaleLineitem o) {
		return this.quantity.compareTo(o.quantity);
	}


}
```

DataManagementAdapter.java

```
package TechnicalSupport;

import java.util.Iterator;
import java.util.Set;
import java.util.SortedSet;
import java.util.TreeSet;

import FoodCourt.FoodCatalog;
import FoodCourt.FoodCourt;
import FoodCourt.FoodDescription;
import FoodCourt.Kitchen;
import Output.NumberTicket;

public class DataManagementAdapter {
    public void loadFoodCourtData(int max_food_number, int max_kitchen_number, FoodCourt
foodcourt) {
        FoodCatalog fc;
        int tmp;

        String[] food_name = new String[max_food_number];
        String[] food_price = new String[max_food_number];
```

```
String[] kitchen_name = new String[max_food_number];

FileManager.loadCategoryFile(kitchen_name, max_kitchen_number);

Set<Kitchen> kitchens = foodcourt.getKitchenSet();

Kitchen k;

for(int i =0; i < max_kitchen_number; i++) {
        if(kitchen_name[i] == null) break;
        fc = new FoodCatalog(kitchen_name[i]);
        foodcourt.getAllFoodCatalog().add(fc);
        k = new Kitchen(fc);
        kitchens.add(k);
}

FileManager.loadFoodFile(kitchen_name, food_name, food_price, max_food_number);
for(int i =0; i< max_food_number; i++) {
        if(food_name[i] == null) break;
        fc = foodcourt.getFoodCatalog(kitchen_name[i]);
        tmp = Integer.parseInt(food_price[i]);
        fc.makeFoodDescription(food_name[i], tmp);
}
```

```
    }

    public void saveFoodCourtData(int max_food_number, int max_kitchen_number,
SortedSet<FoodCatalog> fc) {

        Iterator<FoodCatalog> cit = fc.iterator();

        Iterator<FoodDescription> dit;

        FoodCatalog cur = null;

        FoodDescription fd = null;

        int index = 0;

        int inner_index = 0;

        String[] kitchen_name = new String[max_kitchen_number];

        String[] inner_kitchen_name = new String[max_food_number];

        String[] food_name = new String[max_food_number];

        String[] food_price = new String[max_food_number];

        SortedSet<FoodDescription> fooddescriptions = new TreeSet<FoodDescription>();

        while(cit.hasNext() && index < max_kitchen_number) {

                cur = cit.next();

                kitchen_name[index] = cur.getKitchen_name();

                fooddescriptions = cur.getAllFooddescriptions();
```

```
                dit = fooddescriptions.iterator();
                while(dit.hasNext() && inner_index < max_food_number) {
                        fd = dit.next();
                        food_name[inner_index] = fd.getFood_name();
                        food_price[inner_index] = String.format("%d", fd.getPrice());
                        inner_kitchen_name[inner_index] = kitchen_name[index];
                        inner_index++;
                }

                index++;
        }

        FileManager.saveCategoryFile(kitchen_name, max_kitchen_number);
        FileManager.saveFoodFile(inner_kitchen_name, food_name, food_price,
max_food_number);
    }
}
```

FileManager.java

```
package TechnicalSupport;



import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;

public class FileManager
{
	private static   String file_directory = "Data";
	private static String file_of_food = "Food.dat";
	private static String file_of_kitchen = "kitchen.dat";

	private static void makeNewDirectoryNFile()
	{
		try
```

```
{
		File folder = new File(file_directory);
		if(!folder.exists())
		{
				folder.mkdir();
		}

		File file = new File(file_directory + "₩₩" + file_of_food);
		PrintWriter print_writer = new PrintWriter(new BufferedWriter(new
FileWriter(file)));
		print_writer.println("노루향₩t짜장면₩t6000");
		print_writer.println("노루향₩t짬뽕₩t7000");
		print_writer.println("노루향₩t탕수육₩t12000");
		print_writer.println("팔매₩t된장찌개₩t8000");
		print_writer.println("팔매₩t막국수₩t4000");
		print_writer.println("팔매₩t순두부찌개₩t7000");
		print_writer.println("스페셜쉐프₩t스테이크₩t9000");
		print_writer.println("스페셜쉐프₩t스파게티₩t8000");
		print_writer.println("CongCong₩tnoodle₩t9000");
		print_writer.println("CongCong₩trice₩t8000");
		print_writer.close();
		//딱 한번 테스트를 위한 만들어지는 초기 메뉴들입니다.
		file = new File(file_directory + "₩₩" + file_of_kitchen);
```

```
                print_writer = new PrintWriter(new BufferedWriter(new FileWriter(file)));
                print_writer.println("노루향");
                print_writer.println("팔매");
                print_writer.println("스페셜쉐프");
                print_writer.println("CongCong");
                print_writer.close();

                System.out.println("데이터 폴더가 손상되어, 새로 생성하였습니다. 프로그램을 다시 시작해주세요.");
                System.exit(0);
            }
            catch(Exception ee)
            {

            }
        }

    //파라미터로 들어온 kitchen_name과 food_name, food_price를 파일로부터 읽어와서 줍니다. max_food_number에 배열의 크기를 입력해주어야합니다.
    //예를들어 중국집의 6000원짜리 짜장면은 kitchen_name[0] = "중국집" food_name[0] = "짜장면" food_price[0] = "6000"으로 반환될 것입니다.
    public static void loadFoodFile(String[] kitchen_name, String[] food_name, String[] food_price, int max_food_number )
    {
```

```
try
{
	int index = 0;
	File file = new File(file_directory + "₩₩" + file_of_food);
	FileReader file_reader = new FileReader(file);
	BufferedReader buffered_reader = new BufferedReader(file_reader);
	String tmp = ""; String[] splited_string = new String[3];

	while(((tmp = buffered_reader.readLine()) != null)&&(index < max_food_number))
	{
		splited_string = tmp.split("₩t");
		kitchen_name[index] = splited_string[0];
		food_name[index] = splited_string[1];
		food_price[index] = splited_string[2];

		index++;
	}

	buffered_reader.close();
}
catch (FileNotFoundException e)
{
	try
```

```
                {
                        makeNewDirectoryNFile();
                }
                catch(Exception ee)
                {

                }
        }
        catch(IOException e)
        {
                System.out.println(e);
        }
    }

    //파라미터로 입력된 kitchen_name에 파일의 내용을 불러서 반환해줍니다. max_kitchen에 배열의 크기를 입력해야 합니다.
    //예를들어 중식을 다루는 중국집은 kitchen_name[0] = "중국집"과 같이 반환될 것입니다.
    public static void loadCategoryFile(String[] kitchen_name, int max_kitchen_number)
    {
        try
        {
                int index = 0;
                File file = new File(file_directory + "₩₩" + file_of_kitchen);
```

```
                FileReader file_reader = new FileReader(file);
                BufferedReader buffered_reader = new BufferedReader(file_reader);
                String tmp = "";

                while(((tmp = buffered_reader.readLine()) != null)&&(index < max_kitchen_number))
                {
                        kitchen_name[index] = tmp;
                        index++;
                }

                buffered_reader.close();
                file_reader.close();
        }
        catch (FileNotFoundException e)
        {
                makeNewDirectoryNFile();
        }
        catch(IOException e)
        {
                System.out.println(e);
        }
    }
```

```
//파라미터로 들어온 주방명, 음식이름, 음식가격의 쌍을 저장합니다.
public static void saveFoodFile(String[] kitchen_name, String[] food_name, String[] food_price, int max_food_number )
{
	try
	{
		File folder = new File(file_directory);
		if(!folder.exists())
		{
			folder.mkdir();
		}

		int index = 0;
		File file = new File(file_directory + "₩₩" + file_of_food);
		PrintWriter print_writer = new PrintWriter(new BufferedWriter(new FileWriter(file)));

		while(index < max_food_number&&(kitchen_name[index] != null))
		{
			print_writer.printf("%s₩t%s₩t%s₩n", kitchen_name[index], food_name[index], food_price[index]);
			index++;
		}
```

```
                print_writer.close();
        }
        catch (FileNotFoundException e)
        {
        }
        catch(IOException e)
        {
        }
}

//파라미터로 들어온 주방명을 저장합니다.
public static void saveCategoryFile(String[] kitchen_name, int max_kitchen_number)
{
        try
        {
                File folder = new File(file_directory);
                if(!folder.exists())
                {
                                folder.mkdir();
                }

                int index = 0;
```

```
                File file = new File(file_directory + "₩₩" + file_of_kitchen);
                PrintWriter print_writer = new PrintWriter(new BufferedWriter(new
FileWriter(file)));

                while((index < max_kitchen_number )&&(kitchen_name[index] != null))
                {
                        print_writer.printf("%s₩n", kitchen_name[index]);
                        index++;
                }

                print_writer.close();
        }
        catch (FileNotFoundException e)
        {
        }
        catch(IOException e)
        {
        }
    }


}
```