# 편의점 POS 기능 구현

- 팀 이름: Team 12

- 팀원: 박정현 박현진 서은서 최수아 황인영

- Level: L1

# 목차

# 1. 서론

## 1) 팀 정보

- 팀 이름: Team 12
- 팀원: 박정현(20205602) 박현진(20186756) 서은서(20206147) 최수아(20183611) 황인영(20213865)
- Level: L1

## 2) 과제 주제

"편의점 POS기의 기능 구현"

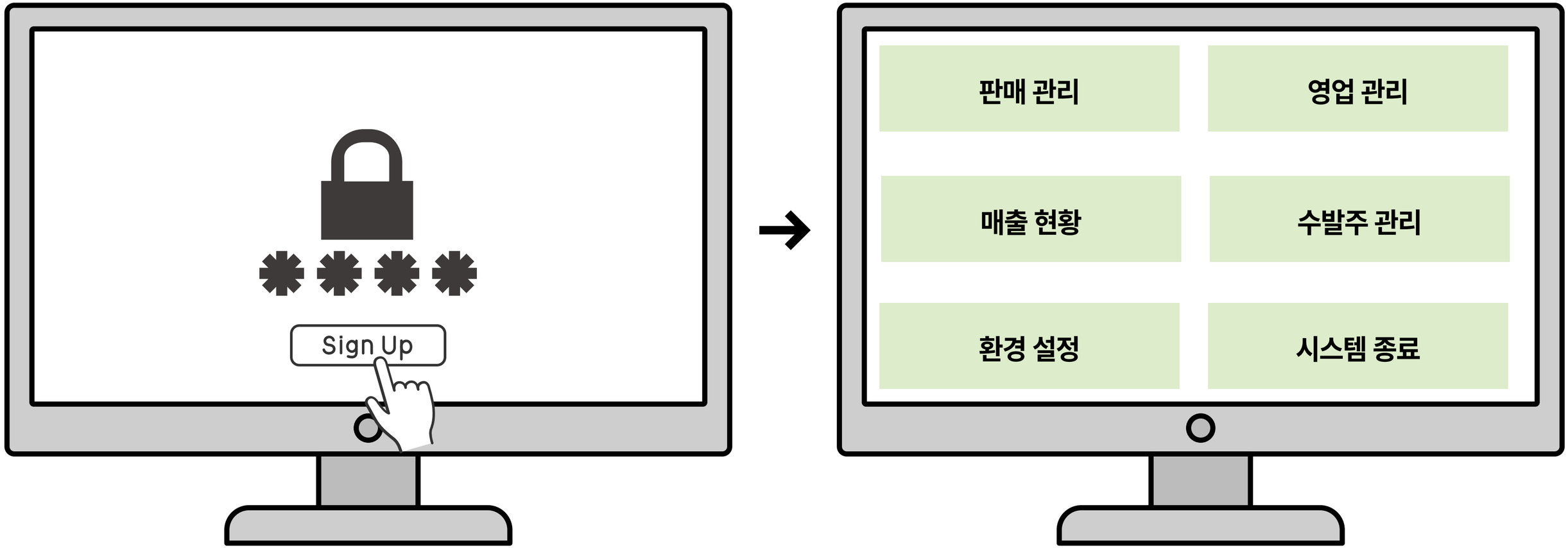
## 3) 프로젝트 목표

"실제 편의점의 POS에서의 기능을 객체지향의 개념을 이용하여 cpp로써 구현하고자 한다."

## 4) 설계 범위 지정

- 판매 관리 + 영업 관리 (마감 정산, 준비금 등록, 수발주 관리)

# 2. 편의점 POS 기능 설명 POS기의 시작 화면



# 2. 편의점 POS 기능 설명 1) 실제 편의점 POS기 - 판매관리

## 판매 관리 : 물건 계산의 기능

▶ POS기에 등록 된 상품버튼을 누르면 구매 목록에 추가한다.

▶ [ + ] / [ - ] 버튼을 이용하여 수량의 증가/감소 조절가능 하다.

▶ 전체 취소 버튼을 이용해 구매목록 초기화시킨다.

▶ 결제 방식 : 카드 & 현금 & 복합 결제(현금 + 카드)

- 카드 결제 : 카드 결제 버튼을 누른 후 IC 칩 방향으로 카드를 삽입한다.

  If 삼성페이로 결제 시 , 카드리더기 옆에 가져다 대면 결제가 가능하다.

- 현금 결제 : 현금 결제 버튼을 누른 후 손님에게 받은 돈을 금고에 저장

  → 거스름돈 발생 시 금고에서 돈을 꺼내 손님에게 거슬러준다.

  If 손님이 현금영수증 요청 시, 전화번호를 직접 입력 혹은 입력 요청을 받아 현금영수증을 발행한다.

- 복합 결제 : 카드 결제와 현금 결제의 방법을 혼합하여 사용한다.

  결제할 금액을 입력 한 후 결제 방법을 선택하면 위의 두 방식과 동일한 방법으로 결제가 진행된다.

+) 할인 처리 : 할인버튼을 누르면 퍼센트 할인 혹은 금액 할인 중에 선택 하여 할인을 적용할 수 있다.

# 2. 편의점 POS 기능 설명 1) 실제 편의점 POS기 - 판매관리

## 판매 관리 : 물건 계산의 기능

▶ 반품 처리

- 첫번째 방식 : 영수증 관리 → 날짜 선택 → (저널) 조회 선택 후 결제 기록을 조회하여 환불을 진행한다.
- 두번째 방식 : 영수증 하단에 존재하는 바코드를 찍어 환불을 진행한다.
- 복합 결제로 결제했을 시 환불 또한 복합으로 받아야한다.

  ex) 여러장의 카드로 결제했다면, 환불을 위해서 결제에 사용했던 모든 카드가 필요하다.

▶ 결제 및 환불 진행 시 총 5만원 이상 계산 시 손님의 서명을 받아야 한다.

▶ 물품에 바코드가 없는 경우 '싯가버튼'을 이용하여 직접 값을 입력할 수 있다.

# 2. 편의점 POS 기능 설명 1) 실제 편의점 POS기 - 마감 정산 및 기타

## 마감 정산

▶ 마감 정산버튼을 누를 시 자동으로 금일 매출이 포스기에 기록된다.

## 기타

▶ 교통카드 충전

▶ 편의점 택배 결제

▶ 공병 회수

▶ 폐기 관리

# 2. 편의점 POS 기능 설명 2) 우리가 만드는 POS기

## 실제 POS기 기능들

▶ 판매 관리

▶ 영업 관리 : 마감 정산, 준비금 등록, 수발주 관리


## 우리의 POS기에 들어갈 기능들

▶ 판매 관리(상품 판매 관련 기능)

▶ 환불 관리(상품 환불 관련 기능)

▶ 재고 관리(편의점 내부 재고 관리 기능)

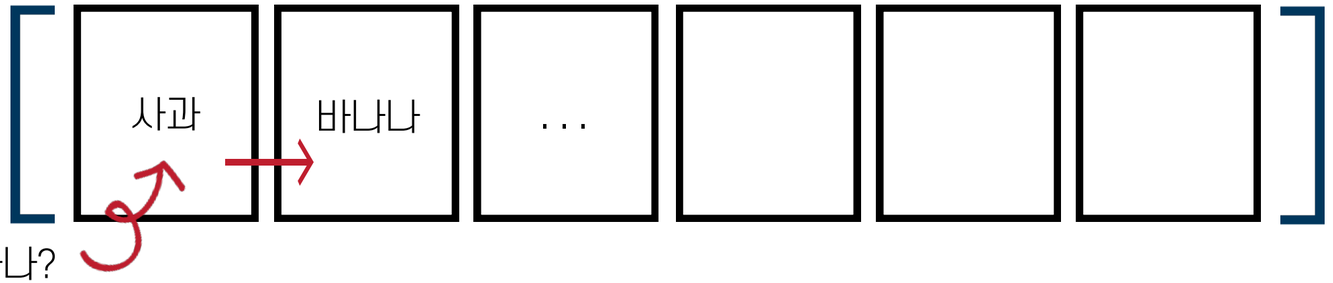
▶ 회계 관리(마감 정산, 준비금 등록, 금고 관리, 자본금 관리 등)

# 2. 편의점 POS 기능 설명 2) 우리가 만드는 POS기 - 판매 관리(purchase_management)

## 판매 관리 : 물건 계산의 기능

▶ 우리의 프로그램은 품목들의 정보가 배열(array)로 이루어져 있고, 배열(array)내에서의 검색을 통해 품목을 찾을 수 있다.

ex)




* 배열에 담긴 각 품목은 item 객체

▶ 장바구니에 상품추가 : 품목의 이름을 직접 입력하여 추가한다.

만약 배열에 들어있는 품목이 아니라면, 정보를 새로 입력하여 품목 배열에 추가할 수 있다.

▶ 구매가 확정 될 시 purchase 객체(구입 날짜, 구입하는 품목들, 총 가격, 총 개수)가 판매 기록(purchase_log)에 저장된다.

*purchase는 결제 영수증의 역할을 한다.

▶ 판매 기록에 저장될 때 재고(item_list)에 있는 해당 품목의 개수가 줄고 자본금이 증가한다.

# 2. 편의점 POS 기능 설명 2) 우리가 만드는 POS기 - 환불

프로그램 작동 방식을 생각하면, 바코드를 찍어 환불하는 것과
영수증 기록을 purchase_id로 조회해 환불하는 방법의 구분이 불분명 하기에 하나로 통합해서 구현

## 환불 관리

▶ POS의 결제 기록은 판매 기록(purchase_log)에 남아있다.

▶ 판매 기록에 내장된 영수증 번호를 이용하여 결제 내역을 조회 후 환불을 진행한다.

▶ 환불 진행 시 재고가 환불된 개수만큼 증가하고 자본금이 줄어든다.

▶ 환불이 완료되면 해당 판매 기록이 '환불완료' 상태를 갖는다.

# 2. 편의점 POS 기능 설명 2) 우리가 만드는 POS기 - 재고 관리 / 회계 관리 / 기타

## 재고 관리

▶ 각 물품의 유통기한을 파악하여 유통기한이 지난 물품들을 마감 정산과 동시에 폐기처분한다. (매일 00시에 진행)

→ 이때, 유통기한의 비교는 *item_exdate* class에서만 이루어진다.

▶ 물품의 재고가 부족한 경우  해당 물품을 구입하여 재고를 채운다.

→ 이때, 재고의 수는 증가하고 자본금은 줄어든다.

## 회계 관리

▶ 현금 매출과 카드 매출을 분리하여 관리한다.

▶ 편의점 마감 시(dayclose), 당일 총 매출을 일일 매출 목록(sales_log[오늘 날짜: 매출액])에 추가한다.

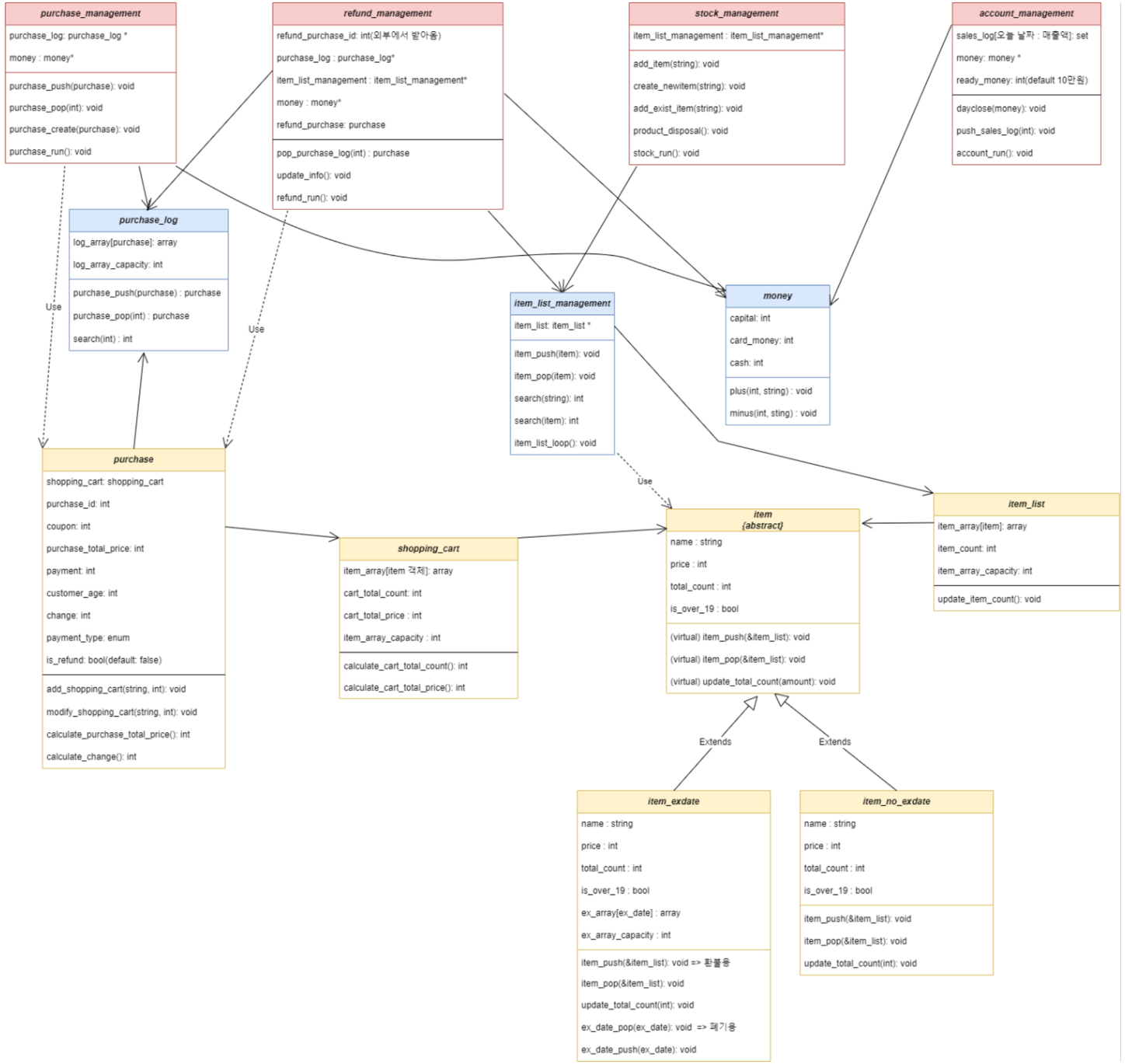
▶ 준비금(거스름돈을 위해 일정 금액을  빼놓은 금액) : 10만원

→ 날이 바뀔 때마다 10만원으로 다시 갱신된다.

## 기타(제한사항)

▶ 교통 카드 충전 기능, 택배 결제, 공병회수, 5만원이상 결제 혹은 환불 시 사인을 받은 과정은 구현에서 제외하였다.

# 3. UML


purchase_management
purchase_log: purchase_log *
money : money*
purchase_push(purchase): void
purchase_pop(int): void
purchase_create(purchase): void
purchase_run(): void
refund_management
refund_purchase_id: int(외부에서 받아옴)
purchase_log : purchase_log*
item_list_management : item_list_management*
money : money*
refund_purchase: purchase
pop_purchase_log(int) : purchase
update_info(): void
refund_run(): void
stock_management
item_list_management : item_list_management*
add_item(string): void
create_newitem(string): void
add_exist_item(string): void
product_disposal(): void
stock_run(): void
account_management
sales_log[오늘 날짜 : 매출액]: set
money: money *
ready_money: int(default 10만원)
dayclose(money): void
push_sales_log(int): void
account_run(): void
purchase_log
log_array[purchase]: array
log_array_capacity: int
purchase_push(purchase) : purchase
purchase_pop(int) : purchase
search(int) : int
Use
Use
item_list_management
item_list: item_list *
item_push(item): void
item_pop(item): void
search(string): int
search(item): int
item_list_loop(): void
money
capital: int
card_money: int
cash: int
plus(int, string) : void
minus(int, sting) : void
purchase
shopping_cart: shopping_cart
purchase_id: int
coupon: int
purchase_total_price: int
payment: int
customer_age: int
change: int
payment_type: enum
is_refund: bool(default: false)
add_shopping_cart(string, int): void
modify_shopping_cart(string, int): void
calculate_purchase_total_price(): int
calculate_change(): int
Use
shopping_cart
item_array[item 객체]: array
cart_total_count: int
cart_total_price : int
item_array_capacity : int
calculate_cart_total_count(): int
calculate_cart_total_price(): int
item
{abstract}
name : string
price : int
total_count : int
is_over_19 : bool
(virtual) item_push(&item_list): void
(virtual) item_pop(&item_list): void
(virtual) update_total_count(amount): void
item_list
item_array[item]: array
item_count: int
item_array_capacity: int
update_item_count(): void
Extends
Extends
item_exdate
name : string
price : int
total_count : int
is_over_19 : bool
ex_array[ex_date] : array
ex_array_capacity : int
item_push(&item_list): void => 환불용
item_pop(&item_list): void
update_total_count(int): void
ex_date_pop(ex_date): void => 폐기용
ex_date_push(ex_date): void
item_no_exdate
name : string
price : int
total_count : int
is_over_19 : bool
item_push(&item_list): void
item_pop(&item_list): void
update_total_count(int): void

# 3. UML

## class 설명

1. main()에 생성(객체화)되는 아래 4가지 class들을 통해 나머지 class에 접근한다.
    - purchase_management => 판매 관리
    - refund_management => 환불 관리
    - stock_management => 재고 관리
    - account_management => 회계 관리

=> management 객체들을 통해서만 특정 기능 수행이 가능하다.

2. 아래 3가지 class는 main() 생성(객체화)되기는 하지만 직접적으로 쓰이지는 않는다. (위 4가지 class 내부에 포함되어 사용됨)
    - item_list_management
    - purchase_log
    - money

3. 나머지 class들은 management class 내부에서 생성 또는 사용된다.

## 추가 설명

main()에서 각 기능을 수행(test)하지 않고
management 객체들의 run() 함수를 통해서 수행(test)한다.
=> main()에서는 판매관리/환불관리/재고관리/회계관리 중 무엇을 할지만 선택하면 된다.

# 3. UML class 설명

## item_list_management

| 이름 | 형식 | 설명 |
|---|---|---|
| item_list | item_list* | item_list 객체를 가리키는 포인터 |
| | | |
| item_push(item) | void | item_list에 item 추가<br>→ item객체의 push 메서드 이용 |
| item_pop(item) | void | item_list에 item 삭제<br>→ item객체의 pop 메서드 이용 |
| search(string) | int | item의 string(name)받아서 item_list의객체에 접근 |
| search(item) | int | 인자로 받은 item의 이름과 같은 item을 item_array에서 search<br>→ 성공하면 item_list에서 index 번호 return / 실패하면 -1 반환 |
| item_list_loop() | void | item_list에 담긴 모든 item 객체들에 차례로 접근 |

# 3. UML class 설명

## item_list

| 이름 | 형식 | 설명 |
|---|---|---|
| item_array[item] | array | item 객체들이 담김 |
| item_count | int | item_array에 담긴 item 수 |
| tem_array_capacity | int | item_array의 capacity |
| | | |
| update_item_count() | void | item_array에 담긴 item 수 update |

# 3. UML class 설명

## item (abstract class)

| 이름 | 형식 | 설명 |
|---|---|---|
| name | string | item의 이름 |
| price | int | item의 가격 |
| total_count | int | item의 총 개수 |
| is_over_19 | bool | 19세 미만 판매 가능 여부 |
| | | |
| (virtual) item_push(& item_list) | void | item_list안에 본인(this) 추가 |
| (virtual) item_pop(& item_list) | void | item 객체 새로 생성 및 복사 (ex_date는 필요한 일부만 추출) |
| (virtual) update_total_count(int) | void | item_class 내부의 총 수량 update |

# 3. UML class 설명

## item_exdate (sub class)

| 이름 | 형식 | 설명 |
|---|---|---|
| name | string | item의 이름 |
| price | int | item의 가격 |
| total_count | int | item의 총 개수 |
| is_over_19 | bool | 19세 미만 판매 가능 여부 |
| ex_array[ex_date] | array | ex_date 형식 : 20221122 |
| ex_array_capacity | int | ex_array의 capacity |
| | | |
| item_push(&item_list) | void | item_list안에 본인(this) 추가 |
| item_pop(& item_list) | void | item 객체 새로 생성 및 복사 (ex_date는 필요한 일부만 추출) |
| update_total_count(int) | void | item_class 내부의 총 수량 update |
| ex_date_pop(ex_date) | void | item객체 내부 ex_array에서 pop (폐기시 오늘 날짜가 ex_date로 들어감) |
| ex_date_push(ex_date) | void | ex_date 받아서 item객체 내부 ex_array에 push |

# 3. UML class 설명

## item_no_exdate (sub class)

| 이름 | 형식 | 설명 |
|---|---|---|
| name | string | item의 이름 |
| price | int | item의 가격 |
| total_count | int | item의 총 개수 |
| is_over_19 | bool | 19세 미만 판매 가능 여부 |
| | | |
| item_push(&item_list) | void | item_list안에 본인(this) 추가 |
| item_pop(& item_list) | void | item 객체 새로 생성 및 복사 (ex_date는 필요한 일부만 추출) |
| update_total_count(int) | void | item_class 내부의 총 수량 update |

# 3. UML class 설명

## stock_management

| 이름 | 형식 | 설명 |
|---|---|---|
| item_list_management | item_list_management* | item_list_management를 가리키는 포인터 |
| | | |
| add_item(string) | void | item_list_management의 search 메서드 이용해서<br>이미 존재하면→ add_exist_item()<br>존재하지 않으면 → create_newitem() |
| create_new_item(string) | void | item_list에 존재하지 않는 item이라면 관련 정보 입력 받은 후 새로 item 객체 생성해서 item_list에 add |
| add_exist_item(string) | void | 이미 item_list에 존재하는 item이라면 유통기한만 입력 받아서 item객체에 add |
| product_disposal() | void | dayclose 할 때 item_list에 접근하여 상품 폐기 |
| stock_run() | void | 재고 관리 실행(test) |

# 3. UML class 설명

## shopping_cart

| 이름 | 형식 | 설명 |
|---|---|---|
| item_array[item] | array | 손님이 구매하는 모든 item 객체들이 담기는 array |
| cart_total_count | int | 손님이 구매한 모든 물건들의 총 개수 |
| cart_total_price | int | 손님이 구매한 모든 물건들의 총 가격 |
| item_array_capacity | int | item_array의 capacity |
| | | |
| calculate_cart_total_count(void) | int | 손님이 구매하는 모든 물건의 총 수량 계산 |
| calculate_cart_total_price(void) | int | 손님이 구매하는 모든 물건의 총 가격 계산 |

# 3. UML class 설명

## purchase_management

| 이름 | 형식 | 설명 |
|---|---|---|
| purchase_log | purchase_log * | purchase_log 객체를 가리키는 포인터 |
| money | money * | money 객체를 가리키는 포인터 |
| | | |
| purchase_push(purchase) | void | 구매 기록을 purchase_log에 집어 넣음 |
| purchase_pop(purchase) | void | 구매 기록을 purchase_log에서 꺼내옴 |
| purchase_create(purchase) | void | 구매 기록을 새롭게 생성 |
| purchase_run() | void | 구매 관리 실행(test) |

# 3. UML class 설명

## purchase_log

| 이름 | 형식 | 설명 |
|---|---|---|
| log_array[purchase] | array | purchase 객체들이 담겨있는 array |
| log_array_capacity | int | log_array의 capacity |
| | | |
| purchase_push(purchase) | purchase | 구매 기록을 log_array에 추가 |
| purchase_pop(int) | purchase | 구매 기록을 log_array에서 pop (영수증 번호 이용해서 search) |
| search(int) | int | 영수증 번호 이용해서 purchase 객체 찾기 |

# 3. UML class 설명

## purchase (=영수증, 구매 기록)

| 이름 | 형식 | 설명 |
|---|---|---|
| shopping_cart | shopping_cart | shopping_cart 객체 |
| purchase_id | int | 영수증 번호 |
| coupon | int | 할인은 항상 원단위(% 할인 없음) |
| purchase_total_price | int | 영수증에 적혀있는 모든 물건의 총 가격 |
| payment | int | 손님이 지불한 금액 |
| customer_age | int | 손님의 나이(19세 미만 판매 금지 상품 구매 가능 여부 판단) |
| change | int | 거스름돈(카드일 경우 default = 0) |
| payment_type | enum | 카드 or 현금 |
| is_refund | bool | default : FALSE => 환불 진행시 TRUE로 바꿈 |
| | | |
| add_shopping_cart(string, int) | void | shopping cart에 item add (이름, 수량) |
| modify_shopping_cart(string, int) | void | shopping cart 수정 (이름, 수량) |
| calculate_purchase_total_price(void) | int | 영수증의 총 가격 계산 |
| calculate_change(void) | int | change 계산 (total price - payment) |

# 3. UML class 설명

## refund_management

| 이름 | 형식 | 설명 |
|---|---|---|
| refund_purchase_id | int | 외부입력 |
| purchase_log | purchase_log* | purchase_log 객체에 접근하는 포인터 |
| item_list_management | item_list_management* | item_list_management 객체에 접근하는 포인터 |
| money | money* | money 객체에 접근하는 포인터 |
| refund_purchase | purchase | instance of purchase (구매 기록) => 환불 진행마다 새롭게 저장 |
| | | |
| pop_purchase_log(int) | purchase | 환불 진행에 필요한 해당 구매기록 pop (영수증 번호 이요) |
| update_info(void) | void | 관련 다른 객체들 정보 update (purchase_log, item_list, money 등)<br>=> 환불 도중에 다른 class update하지 않고 환불 완료가 되면 한꺼번에 update하기 위해 만듦 |
| refund_run(void) | void | 환불 관리 실행(test) |

# 3. UML class 설명

## money

| 이름 | 형식 | 설명 |
|---|---|---|
| capital | int | 자본금 |
| card_money | int | 카드 |
| cash | int | 현금 |
| | | |
| plus(int, string) | void | 현금 / 카드 plus (money, type(현금/카드)) |
| minus(int, string) | void | 현금 / 카드 minus (money, type(현금/카드)) |

# 3. UML class 설명

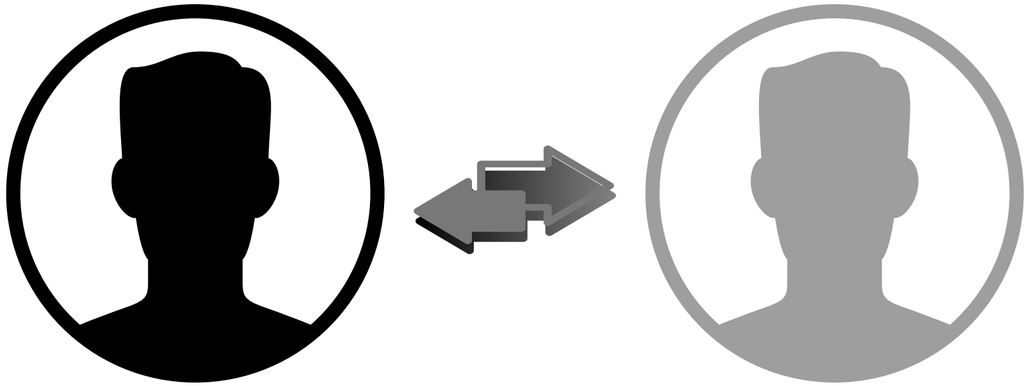
## account_management

| 이름 | 형식 | 설명 |
|---|---|---|
| sales_log[오늘 날짜:매출액] | set | 오늘 매출액 담겨있는 array |
| money | money* | money 객체에 접근하는 포인터 |
| ready_money | int | dafault 100,000 |
| | | |
| dayclose(money) | void | money의 하루 매출액 계산해서 capital 정보 업데이트, sales_log에 추가 |
| push_sales_log(int) | void | sales_log에 하루매출액 push |
| account_run() | void | 회계 관리 실행(test) |

# 4. UML을 이용한 행동분석

편의점 POS 사용자인 매장의 관리자 입장에서 여러 상황들에 대한 행동을 가정하고, 그 행동을 바탕으로 POS기의 기능을 분석하였다.
POS기의 가장 핵심적인 기능인 판매/환불/재고/회계 위주로 분석하였다.

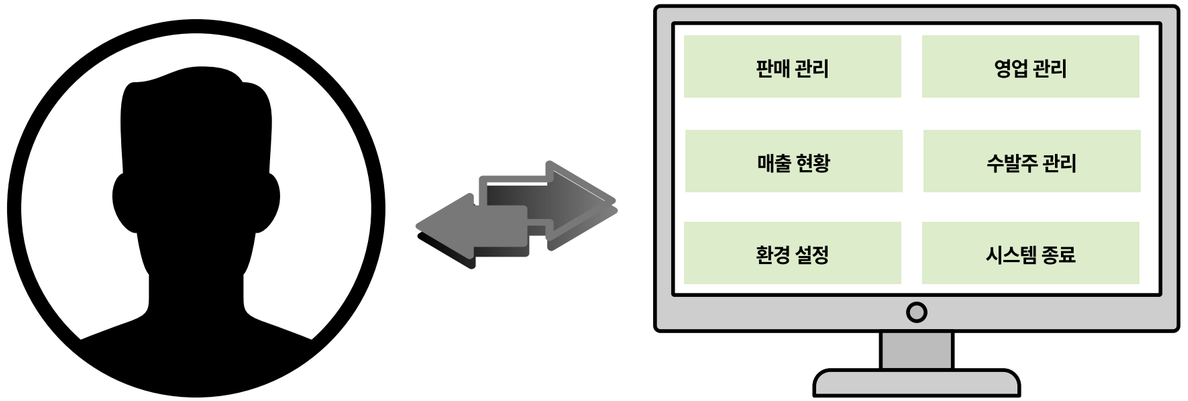
관리자 손님



## 1) 손님과의 상호작용

손님이 들어왔을 때, 손님의 편의점에 대한 용건과 관련된 상황들이다.

관리자




## 2) 관리자의 포스기 사용

관리자의 편의점 상품 발주와 같은 재고 관리나 회계관리 같은
전체적인 편의점 관리와 관련된 상황이다.

# 4. UML을 이용한 행동분석 1-1) 손님이 상품을 구매할 때의 행동 분석

**＊결제 과정 도중에 구매 취소하는 상황 없다고 가정**

=> 만약 취소하고 싶을 경우 결제 완료 후, 환불 통해서만 가능

=> 취소하는 상황이 어떤 단계에서 발생할지 모호하여 매번 확인하기 까다로움(ex. 손님에게 정보 받아오는 단계, 거스름돈 계산단계 등)

## 1. 손님에게서 4가지 정보 받아온다. (purchase 객체 생성)

▶ shopping cart

- add_shopping_cart()를 이용해 손님이 구매할 물건의 종류와 수량을 입력 받는다. (종류가 다를 경우 나눠서 여러번 입력)
- item_list에서 해당 item을 찾고 item copy(새롭게 객체 생성 및 정보 복사) 하여 shopping_cart에 저장

  -> 이때, 손님이 구매하는 수량만큼만 유통기한 정보 pop하여 가져온다.

  -> 나머지 정보는 모두 동일, 유통기한 정보만 다름(ex. item_list에 있는 '사과'라는 item은 편의점 내부 모든 사과의 유통기한, shopping_cart의 '사과'라는 item은 손님이 구매하는 사과의 유통기한)
- cart_total_price와 cart_total_count의 계산을 위해 사용된다.

＊shopping_cart에 item을 담을때 손님의 나이를 이용하여 구매가능한 item인지 판단한다. => 구매 불가능할 경우 해당 item은 shopping_cart에 담지 않고 다음 상품을 담는다.

▶ payment type : 지불 방식(현금 또는 카드)

▶ payment : 손님이 지불한 금액

▶ coupon : 할인되는 만큼의 금액

▶ customer_age : 손님의 나이(19세 미만 판매 금지인 품목 판단)

# 4. UML을 이용한 행동분석 1-1) 손님이 상품을 구매할 때의 행동 분석

## 2. 거스름돈 계산

▶ 현금으로 계산(payment_type == 현금) : caculate_change를 통해 change(거스름돈)계산한다.

▶ 카드로 계산(payment_type == 카드) : change(거스름돈)는 0(default 값)이 된다.

＊구현의 편의성을 위해 현금과 카드의 복합결제는 구현에서 제외

## 3. 구매로 인해 (1)구매 기록 (2)재고 (3)포스기 회계상의 변경이 발생한다.

(1) 구매 기록의 변경

▶ purchase_management class를 이용하여 purchase_log class의 log_array (구매 기록)에 기록을 추가한다.

▶ log_array

- 구매에 사용되었던 purchase class의 객체가 들어간다.
- 이전 구매 정보에 대한 내용들을 저장할 수 있다.

→ 포스 구매 기록을 통해 영수증 출력, 환불과 같은 기능을 수행할 수 있다. (환불완료시 is_refund를 True로 변경)

# 4. UML을 이용한 행동분석 1-1) 손님이 상품을 구매할 때의 행동 분석

(2) 재고의 변경

▶ item_list_management class 안에 있는 search(item)를 이용하여 이름이 같은 item을 탐색한다.

- item_exdate의 경우: 편의점 내부(item_list) item의 ex_array에 환불하는 item의 ex_array를 꺼내온다.

(ex_array는 항상 정렬되어 존재하므로 가장 앞의 ex_date를 꺼낸다.) + item_list 내부 해당 item의 수량을 감소 시킨다.

- item_no_exdate의 경우: 편의점 내부 해당 item의 수량만 감소 시킨다.

=> 두 class의 함수 이름은 같고 내부 동작 방식만 다른 것으로, 코드상에서 item class의 종류를 판단할 필요는 없다.

(3) 포스기 회계상의 변경

▶ purchase class의 calculate_purchase_total_price()를 이용하여 총 구매 금액을 계산한다.

▶ purchase_management의 money를 통해 매출액을 update 시킨다.

# 4. UML을 이용한 행동분석 1-2) 손님이 환불을 요청하러 온 상황에서의 행동 분석

**1. 손님이 준 영수증으로부터, refund_management class를 통해 refund_purchase_id (영수증 번호)를 받는다.**

**2. pop_purchase_log(int)를 통해 purchase_log에 접근하여, 해당 영수증 번호를 가진 purchase 객체를 넘겨받는다.**

▶ 이 과정에서 purchase_log class는 영수증 번호와 search(int)를 이용해 해당되는 구매 기록을 검색한다.

▶ search하여 찾은 purchase 객체를 refund_management class의 refund_purchase()로 넘긴다.

▶ purchase_log에 남은 purchase 객체의 is_refund를 true로 바꾼다.

# 4. UML을 이용한 행동분석 1-2) 손님이 환불을 요청하러 온 상황에서의 행동 분석

### 3. 넘겨진 정보를 바탕으로 (1)재고 (2)포스기 회계상의 변경이 발생한다.

(1) 재고의 변경

▶ refund_management class를 통해 넘겨받은 purchase 객체에 담긴 정보로 item_list_management class 의 search()를 이용하여 item을 찾는다.

- item_exdate의 경우: 편의점 내부(item_list) item의 ex_array에 환불하는 item의 ex_array를 넣는다.
(ex_array는 항상 정렬되어 존재하므로 들어갈 index 찾는 과정 필요) + item_list 내부 해당 item의 수량을 증가 시킨다.
- item_no_exdate의 경우: 편의점 내부 해당 item의 수량만 증가 시킨다.

=> 두 class의 함수 이름은 같고 내부 동작 방식만 다른 것으로, 코드상에서 item class의 종류를 판단할 필요는 없다.

(2) 포스기 회계상의 변경

▶ refund_management 속 money에 접근하여 minus(int)를 이용하여 해당하는 환불가만큼 매출액을 감소시킨다.

+) 우리 POS기에서는 부분환불은 불가하며, 부분 환불은 전체 환불 후 부분 구매하는 것으로 대체한다. (구현의 어려움)

# 4. UML을 이용한 행동분석 2-1) 알바생이 POS기를 통해 재고를 추가할 때의 행동분석

상품 발주 후, item_list에 상품을 추가하는 과정이다. → [1-1 재고의 변경]과 [1-2 재고의 변경]의 구체화

**▶ 유통기한 유/무**

- 유통기한이 없는 상품 : name(상품명), price(가격), total_count(상품의 총 가격), is_over_19(성인에게만 판매할 수 있는 상품일 때 1을 나타냄)
- 유통기한이 있는 상품 : name(상품명), price(가격), total_count(상품의 총 가격), is_over_19(성인에게만 판매할 수 있는 상품일 때 1을 나타냄),
ex_array[ ] (같은 상품 종류 내에서 유통기한을 통해 서로 다른 상품임을 인식할 수 있게 한다. ex. 2022년11월 9일 까지인 상품 2개와 2022년 11월 8일까지인 상품 1개는 [20221108,20221109,20221109] 이와 같이 관리된다.)

* ex_array[]의 구조를 다양한 관점으로 생각해 보았으나 위와 같이 선택한 이유는 다음과 같다.

1. [[20221108, 1], [[20221109, 2]]처럼 수량과 함께 관리하는 경우
   - item_list의 접근은 결제/환불/재고관리/마감정산 등 정말 다양한 상황에서 발생하는데 매번 count를 update시켜야 한다면 => 구현에서 count관리가 매우 까다롭다.
   ex) '사과' 2개를 구매하는 경우 20221108 , 20221109를 각각의 리스트에 접근하여 꺼내와야한다. 또, 수량이 0인 list는 아예 삭제시켜주고, 20221109의 수량은 0으로 바꿔줘야 한다. 구현이 매우 까다로움을 알 수 있다.
2. 편의점은 창고라는 공간적 제약사항이 존재한다.
   - item 품목 하나당 그렇게 많은 유통기한이 존재하기 힘들다. (많아봐야 50개로 추정)
   - 따라서 프로그램의 성능이 크게 저하될 것이라고 보기 힘들다. (search 등의 반복문 수행시)
3. 우리가 선택한 구조 사용시 장점
   - item_list에 pop/push 할 때, 항상 유통기한 순으로 정렬되어 있으므로 가장 앞의 수량부터 하나씩 차례로 삭제/추가하기만 하면 된다.
   (환불로 인한 수량 추가시 item_list에서 index 찾는 과정 필요)

# 4. UML을 이용한 행동분석 2-1) 알바생이 POS기를 통해 재고를 추가할 때의 행동분석

**1. 상품은 item_list에 추가하기 위해, 먼저 item_list_management class안에 있는 search(string)을 이용하여 item을 찾는다.**

**2. 재고 추가**

▶ item_list에  item 존재 시

- stock_management class의 add_exist_item()을 이용하여 재고를 추가한다.
  - 유통기한이 있는 상품 : 유통기한을 받아와 재고를 추가한다.
  - 유통기한이 없는 상품 : 개수만 증가시킨다.

▶ item_list에 item 존재하지 않을 시

- creat_newitem()을 통해 item class에 있는 모든 멤버변수의 정보를 받아와 item을 생성하고, item_push,를 통해 item_list에 넣는다.

# 4. UML을 이용한 행동분석 2-2) 알바생이 마감 정산을 할 때 행동분석

알바생은 account_management class의 sales_log를 통해 지난 날짜와 매출액을 확인할 수 있다.

## 1. 마감 정산의 경우, account_management class의 dayclose를 호출한다.

→ 이를 통해 (1) 회계상의 변화 (2) sales_log의 추가 (3) 유통기한 지난 상품 폐기와 같은 변동이 일어난다.

(1) 회계상의 변화

▶ account_management class에서 money class에 접근하여, 일일 매출액을 계산하고, 이를 capital(자본금)에 더하여 자본금 정보를 업데이트 시킨다.

(2) sales_log의 추가

* 일일 매출액: card_money + cash - ready_money

▶ 오늘 날짜와 함께 card_money와 cash를 더해 계산된 일일 매출액을 push_sales_log를 통해 sales_log에 추가한다.

(3) 유통기한 지난 상품 폐기

▶ stock_management class의 product_disposal을 이용하여, 유통기한이 지난 상품들을 제거한다.

## 2. 마감 정산 후 ready_money(영업준비금)을 100,000으로 초기화한다.

▶ 영업준비금은 매일 10만원씩 준비되는 것으로, 손님에게 거스름돈을 주기 위해 준비해두는 것이다.

-END-