**캡스톤 프로젝트**

**dPos 소스분석**

****

**작성자 : 20146518 손창우**

Sequence

* Run(DB관점)

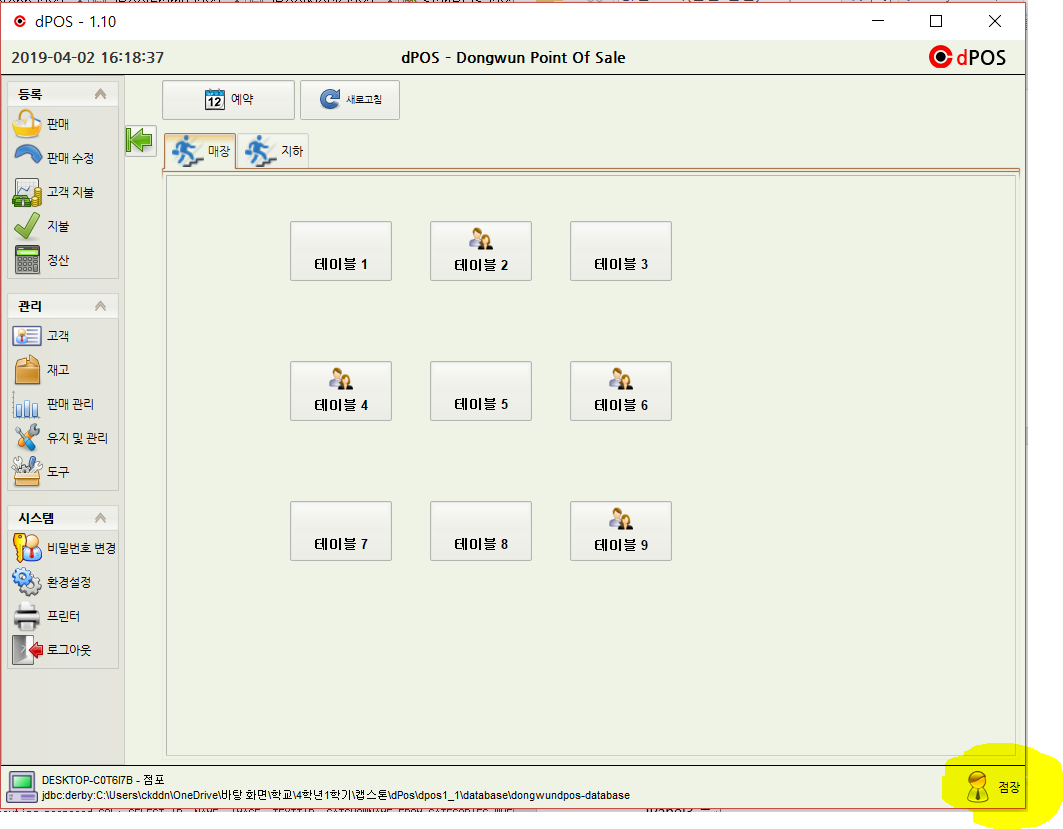
1. PEOPLE테이블에서 사용자 목록 NAME순으로 정렬 (관리자 점장 직원)

* 사용자 버튼 클릭:

1. ID에 따른 Permission을 가져온다. resources테이블에서 content를 가져와서 가동환경 상태를 가져온다.
2. FLOORS테이블에서 매장의 ID, NAME, IMAGE(그림)를 가져온다.
3. PLACES테이블에서 해당 FLOOR에 따른 테이블 정보를 가져온다. (ID NAME X Y CUSTOMER WAITER TICKETID TABLEMOVED)

* Run(Class관점)

1. StartPOS클래스 내 main함수 실행.
2. AppConfig클래스에서 저장되어 있었던 기존 환경을 불러온다.
3. 저장된 화면 모드에 따라 전체화면(JRootKiosk), 창 화면(JRootFrame)을 생성한다. 이후 initFrame함수를 이용해 로그인화면(JRootApp)을 띄워준다.
4. initApp(config)을 통해 이외 다른 주변장치들(ex>프린터) 과도 연결을 수행한다.
5. line.626 listPeopleVisible()을 통해 PEOPLE테이블에 저장된 dPOS사용자들의 정보를 리스트 형태로 가져온다.
6. InitApp함수 이외 closeAppView또는 exitToLogin의 함수의 호출 다음에도showLogin() 이 수행되며 listPeople()을 통해 사용자들을 리스트로 가져와 보여준다.
7. 각 사용자 버튼에 대해 Line.632 new AppUserAction을 등록하여 로그인을 진행할 수 있게 해준다.
8. 로그인 후JPanel3(메인패널 하단)는 로그인 된 사용자를 알려주기 위해 아이콘과 이름을 넣은 label추가하고 메인 패널에 사용자에 대해 생성된 패널(m\_principalapp)을 붙인다



1. 이후 POS화면에서는 각각의 패널들을 이벤트가 일어날 때 만들어주는 것이 아니다. m\_aCreatedViews라는 HashMap<String, JPanel> 자료형에 새로운 작업을 수행할 패널이 만들어 질 때 이들을 저장하며 보여준다. 이전 창으로 돌아가기 용이하게 Stack에도 패널들을 저장하며 이용한다.
2. 사용자 로그인 후 테이블 후불모드는 이에 맞는 패널인 JTicketsBagRestaurantMap을 중간 화면에 보여주게 된다.

POS 분석

1. DB관련 클래스

* Connection: DB 서버와의 연결 유지
* **Statement**: static SQL
* **PreparedStatement**: prepared SQL, 틀이 유지되며 파라미터만 추가적으로 넣어 재사용가능한 쿼리
* **SerializerRead**, **SerializerWrite**: DB에서 읽어오고 쓰는 방식에 대해 정의한다.
* **DataLogic**\*: 각 트랜잭션에 관련된 모든 데이터베이스와의 접근방식을 정의한 클래스이다. 즉, 레코드에 대한 파싱 방법을 매번 명시할 필요 없이 객체를 파라미터로만 전달하여 데이터베이스에 접근(읽기, 쓰기)을 손쉽게 하도록 도와주며 SerializerRead, SerializerWrite를 이용하여 정의한다.

1. 구현관련 클래스 – 테이블 후불 모드 한정

* **Floor**: 매장의 층에 대한 정보를 가지고 있는 클래스.
* **Place**: 테이블에 대한 정보를 가지고 있는 클래스. 해당 테이블이 위치한 층수 같은 정보를 가지고 있으며 각 주문을 식별할 TicketID를 가지고 있다. 이를 정보 통하여 사용중인 테이블과 사용가능한 테이블을 구분한다.

JTicketsBagRestaurantMap클래스에서 Floor와 Place를 관리하며 사용중인 테이블에 대해 아이콘을 표시하는 함수 setPeople도 이 클래스 내부에서 불러지게된다.

* **JTicketsBagRestaurantMap**: 테이블 후불 모드의 Floor와 Place의 위치 지도를 보여주며, 추가적으로 예약기능이 담겨있는 패널. 테이블을 클릭할 때 발생하는 이벤트를 MyActionListener를 통하여 정의하고 있다.

**Line. 198 = pl.getButton().addActionListener(new MyActionListener(pl));**

에서 등록하는 버튼을 클릭할 때 발생할 이벤트를 등록한다.

**MyActionListener 분석** – (Line number는 추가된 코드에 의해 달라질 수 있습니다.)

**m\_PlaceClipboard, ticketclip** 은 테이블 이동시 이동할 테이블의 정보와 티켓 정보 복사하여 담은 변수

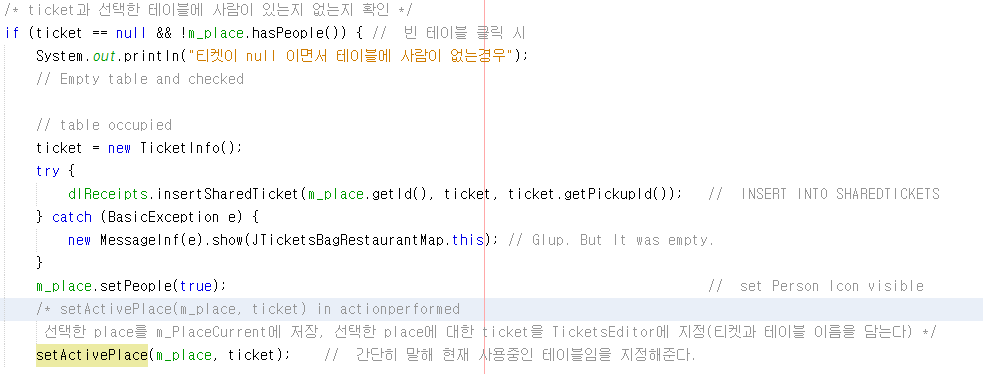
**if (m\_PlaceClipboard == null && customer == null)**

테이블 이동을 누르지 않고 테이블 후불 모드에서 테이블을 선택할 때의 모든 경우에 해당된다.

Ticket은 Place를 통하여 얻는 정보이므로 빈 테이블은 ticket에 대한 정보가 없다

**Case 1: 빈 테이블 클릭 시** Line.582 Ticket == null && !m\_place.hasPeople()

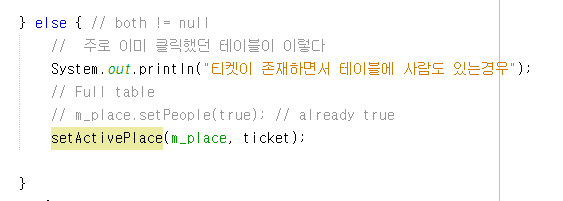
빈 테이블이므로 티켓이 없고 사람도 없는 경우 실행되며 새로운 테이블을 위한 Ticket을 발행한다. **사용중인 테이블의 개수를 증가시킨다.**





**Case 2: 비어 있지 않은 테이블 클릭 시** Line.609 ticket != null && m\_place.hasPeople()

이용중인 테이블이므로 티켓도 가지고 있고 사람도 존재할 때 실행된다. 사용중인 테이블 수에는 변함이 없다.



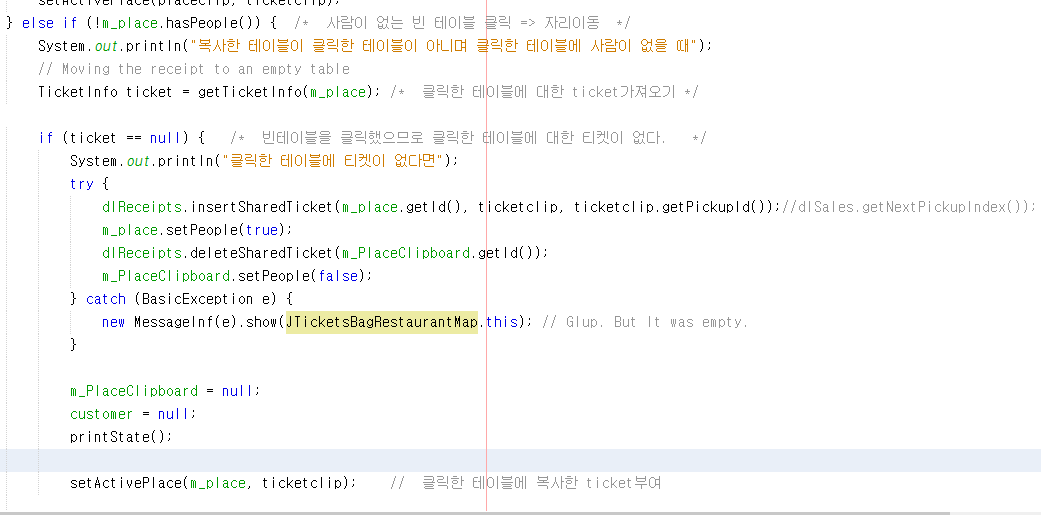
**Case 3: 테이블 이동을 클릭하였고 복사한 위치와 같은 테이블을 클릭할 때**

Line. 670 복사한 테이블의 티켓을 다시 클릭한 위치에 넣어준다. 사용중인 테이블 수에는 변함이 없다.



**Case 4: 테이블 이동을 클릭하였고 복사한 위치와 다른 ‘빈’ 테이블을 클릭할 때**

Line. 678 테이블에 사람이 없을 때(이동), 사용중인 테이블 수에는 변함이 없다.

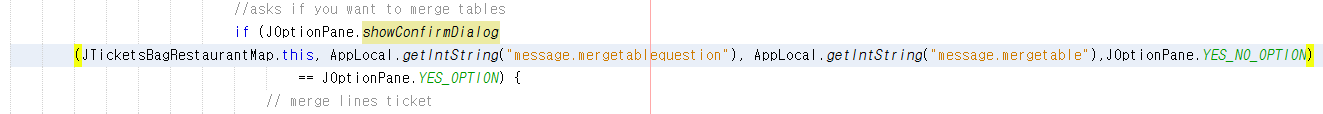


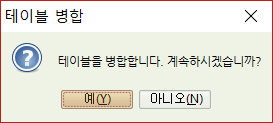
이동하려고 하는 테이블에 정보를 넣은 후 이동하여 빈 테이블에 대한 정보를 제거한다.

dlReceipts 는 DataLogicReceipts 클래스를 참고한다.

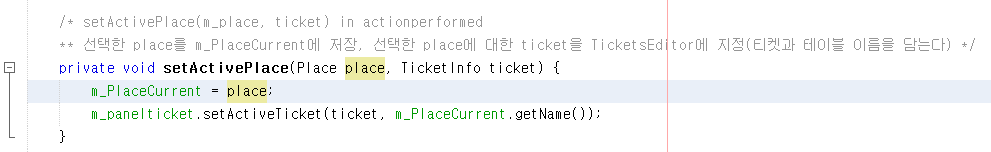
**Case 5: 테이블 이동을 클릭하였고 복사한 위치와 다른 ‘사용중인’ 테이블을 클릭할 때**

Line.707 테이블에 사람이 있을 때(병합), **사용중인 테이블의 개수를 감소시킨다.**



 병합을 물어보는 알림창을 보여줌

* **setAcitvePlace** : case 1, 2의 공통 수행과정, 테이블에 사람을 넣어줄 때 실행된다.
* **setActiveTicket** : m\_panelticket은 상속한 클래스 JTicketBag에서 들고온다



이외 테이블 수의 변화는 계산 후 또는 주문취소 경우에 일어나며 **사용중인 테이블의 개수는 감소한다.**

* **TicketInfo** - JTicketsBagRestaurantMap에서 생성. TicketLineInfo배열을 속성으로 가지고 있다.

ticket, receipt를 엮어서 만든 영수증? 클래스

ID는 랜덤으로 생성한다, 영수증 종류(ex> RECEIPT\_NORMAL), 날짜, 사용자, 고객, attributes 등등 설정

* **JTicketBag-** 모드에 따른 POS기본화면 구성을 결정해준다.

createTicketsBag메서드를 통해 환경설정에서 지정한 모드의 종류

1. 표준모드

JTicketsBagShared 화면을 띄워준다.

1. 테이블 후불 모드

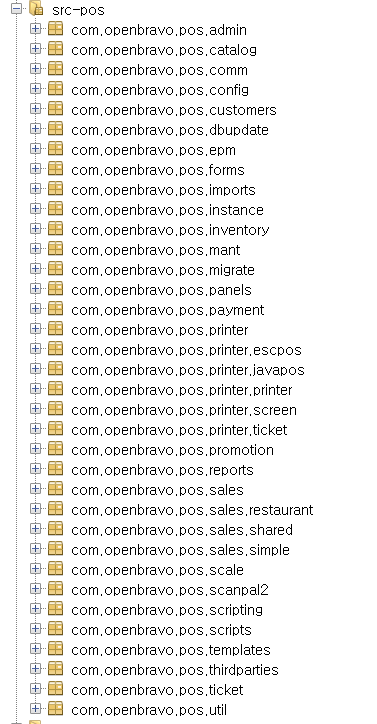
JTicketsBagRestaurantMap 화면을 띄워준다.

우리가 진행할 POS기의 동작화면은 이 클래스를 기준으로 이루어 진다.

1. Default

JTicketsBagSimple 화면을 띄워준다.

SRC-POS 패키지 분석

* ****Package **src-pos**
* Admin: 사원관리, 등록, 역할, 도구 정의 클래스

DB: people, role, resources

* Catalog: 카탈로그, 상품 주문내역 창
* Comm: 프린터와 접속관리 클래스
* Config: POS의 환경설정을 클릭하였을 때의 패널 클래스
* Customer: 고객관리, 고객 예약관리 클래스

DB: customer, reservation

* Dbupdate: DB업데이트…?
* Epm: 휴식과 휴가 관리 클래스

DB: shift, break, leave, shift\_break – dPOS는 운용하지 않는다

* **Forms**: 기본적인 창 구성과 형식이 정의된 클래스 포스기 패널의 시작(StartPOS.java)

Class: StartPOS -> AppConfig(설정) -> JRootKiosk(전체화면), JRootFrame(창 화면) 위 JRootApp(만들어진 화면 위에 시작 화면) -> 사용자 클릭 -> JPrincipalApp(모드에 맞는 화면)위 JTicketsBags(표준모드, 테이블 후불모드) 순으로 실행

* Import: CSV파일로부터 가져오기
* Instance: POS시작시 생성되는 필수 인스턴스
* Inventory
* Mant
* Migrate
* Panels
* Payment
* Printer
* Promotion
* Reports
* Sales: 포스기에서 판매를 위한 클래스

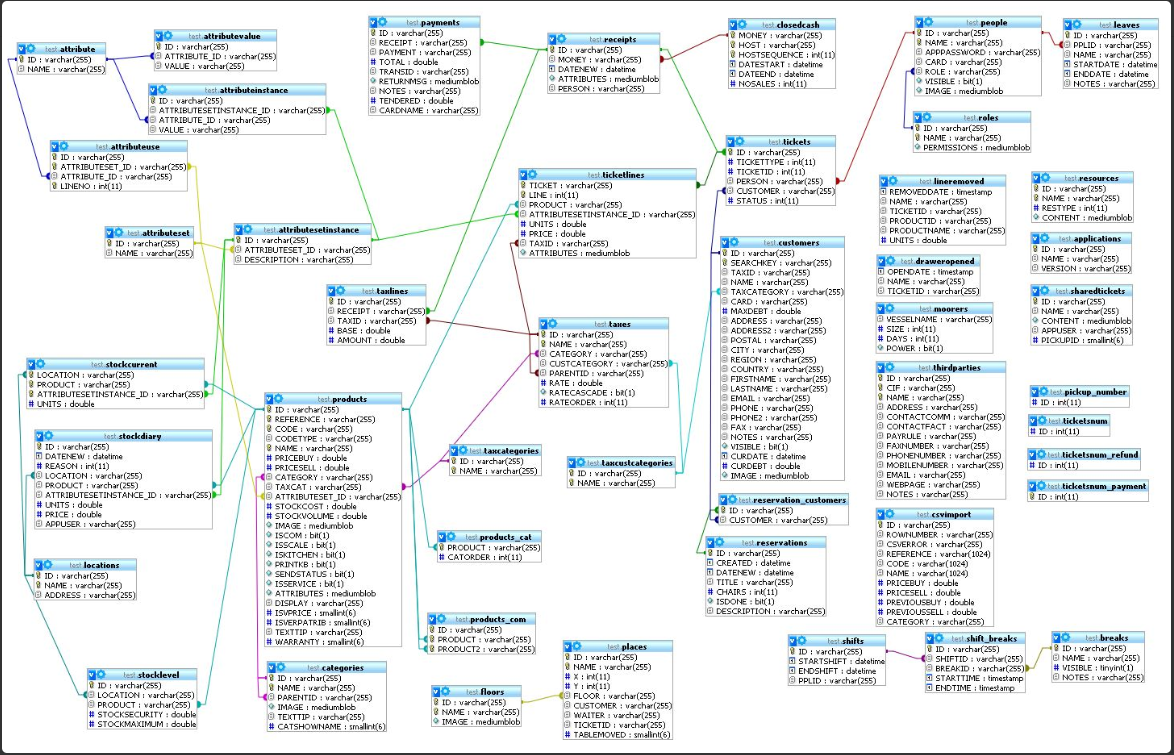
DB: sharedtickets

* + Restaurant: 테이블 후불 모드 – JticketsBagRestarantMap을 실행
  + Shared: 표준모드 – JticketsBagShared를 실행
  + Simple: 일반모드(사용되지 않는다)
* **Sales.restaurant**: 테이블 후불모드를 구성하는 요소들을 담은 클래스
  + **com.ovenbravo.pos.sales.restaurant.leteatgo 패키지추가**
    - **Place의 상태를 담는 클래스 추가 : PlaceStatus.java**
    - **LetEatGo 서버와의 통신을 위한 클래스 추가 : SeatsInfoSender.java**
      * [https://www.java2s.com/Code/Jar/j/Downloadjavajsonjar.htm 에서 java-json.jar.zip](https://www.java2s.com/Code/Jar/j/Downloadjavajsonjar.htm%20에서%20java-json.jar.zip)다운로드 후 속성에서 lib추가, import하기

DB: floor, place

* Scale: 프린터의 기종에 따른 Scale 기능 정의 클래스
* Scanpal2:
* Scripting:
* Scripts: 데이터베이스에 따른 create, upgrade 질의문을 가진 패키지
* Templates: XML파일과(영수증에 쓰여질 형식 정의) png파일을 담고있는 패키지
* Thirdparties: ??
* Ticket: 계산에 따라 나오게되는 ticket에 대한 제어, 정의 클래스
* Util: 포스의 메인 기능이외에 암호화, 인코딩, 카드번호 해석, 라벨, 프린터 등 에 대한 추가적인 기능 정의, 제어 클래스

Database Schema



* TICKETSNUM, TICKETSNUM\_REFUND, TICKETSNUM\_PAYMENT : 거래 일련번호 관리에 사용하는 단일 컬럼 테이블
* closedcash : 정산 관리에 사용하는 테이블, 호스트 단위로 정보를 저장합니다.
* resources : POS 가동 환경을 저장하는 테이블로 content 컬럼이 BLOB으로 XML등 다양한 형태의 정보를 저장합니다.
* localtions : 점포 정보, 초기 DB 생성시 '0', 점포를 생성해 둡니다.
* people : 사용자 정보 테이블, 역할 정보를 담는 roles가 먼저 적절히 설정되어야 합니다.
* roles : XML 기반(BLOB)의 역할 정보 테이블
* leaves, shifts, breaks, shift\_breaks : 휴가/휴식 관련 테이블로 dPOS에서는 관련 기능 사용 안함.
* TAXCATEGORIES : 상품별 세금 분류로 TAXES와 상품 테이블에서 참조
* TAXCUSTCATEGORIES : 고객별 세금 분류로 TAXES와 고객 테이블에서 참조
* TAXES : 세금 기본 정보로 세금 분류를 먼저 설정 해야함
* CATEGORIES : 분류, parentid로 재귀적으로 참조하는 형태
* PRODUCTS : 상품 정보, 분류(CATEGORIES)가 먼저 설정되어야 합니다.
* PRODUCTS\_CAT : 상품 배열 순서 정보, 상품 아이디 기준으로 배열 순서 보관
* SHAREDTICKETS : 티켓 기본 정보- 판매 화면 상단의 새판매/판매보류, 판매취소, 보류목록은 이 테이블을 기준으로 합니다. pickup\_number 테이블로 pickupid(판배번호)를 발급 받음.
  + 결제 시, 취소 시SHAREDTICKETS테이블에서 ID로 검색해 테이블 정보를 삭제한다.
  + 이동 시 존재하던 테이블에서 DELETE FROM, 이동한 테이블 INSERT INTO
* RECEIPTS : 거래 테이블, 판매시 생성. 정산(closedcash) 테이블에 연관해서 생성
* TICKETS : 영수증 테이블, 거래테이블에 연관해서 생성
* TICKETLINES : 개별 주문 내역으로 영수증과 상품 테이블에 연관해서 생성
* STOCKCURRENT : 현재 재고 현황으로 상품, 위치, 속성 테이블과 연관해서 생성
* STOCKLEVEL : 장소별 재고 관리 테이블
* PAYMENTS : 지불 내역으로 거래 테이블과 연관해서 생성
* TAXLINES : 거래단위 세금 정보로 거래 테이블과 연관해서 생성
* DRAWEROPENED : 금전출납기 개방 기록
* LINEREMOVED : 판매 내역에서 삭제할 때 그 내역을 기록하는 테이블
* PRODUCTS\_COM : 보조 상품 내역 관리 테이블로 상품 테이블의 보조 상품 여부 플래그와 함께 상품간 연관성을 관리
* ATTRIBUTESET : 속성 그룹 정의로 각 상품별로 연관 속성은 속성 그룹을 지정합니다.
* ATTRIBUTE : 개별 속성 정의
* ATTRIBUTEUSE : 속성 입력 화면 관리로 속성과 속성 그룹을 연관해서 저장
* ATTRIBUTESETINSTANCE : 속성 그룹의 입력 내역. desceription에 속성 입력 내역 전체를 저장
* ATTRIBUTEINSTANCE : 개별 속성 단위의 입력 내역
* ATTRIBUTEVALUE : 속성별 세부 항목
* CUSTOMERS : 고객 정보 테이블
* Pickup\_number : 좌석 번호 가져오기
* Ticket\_number : 영수증 번호 가져오기
* Thirdparties : 매장정보