헌책 장터 시스템 (Used-book MarketPlace)

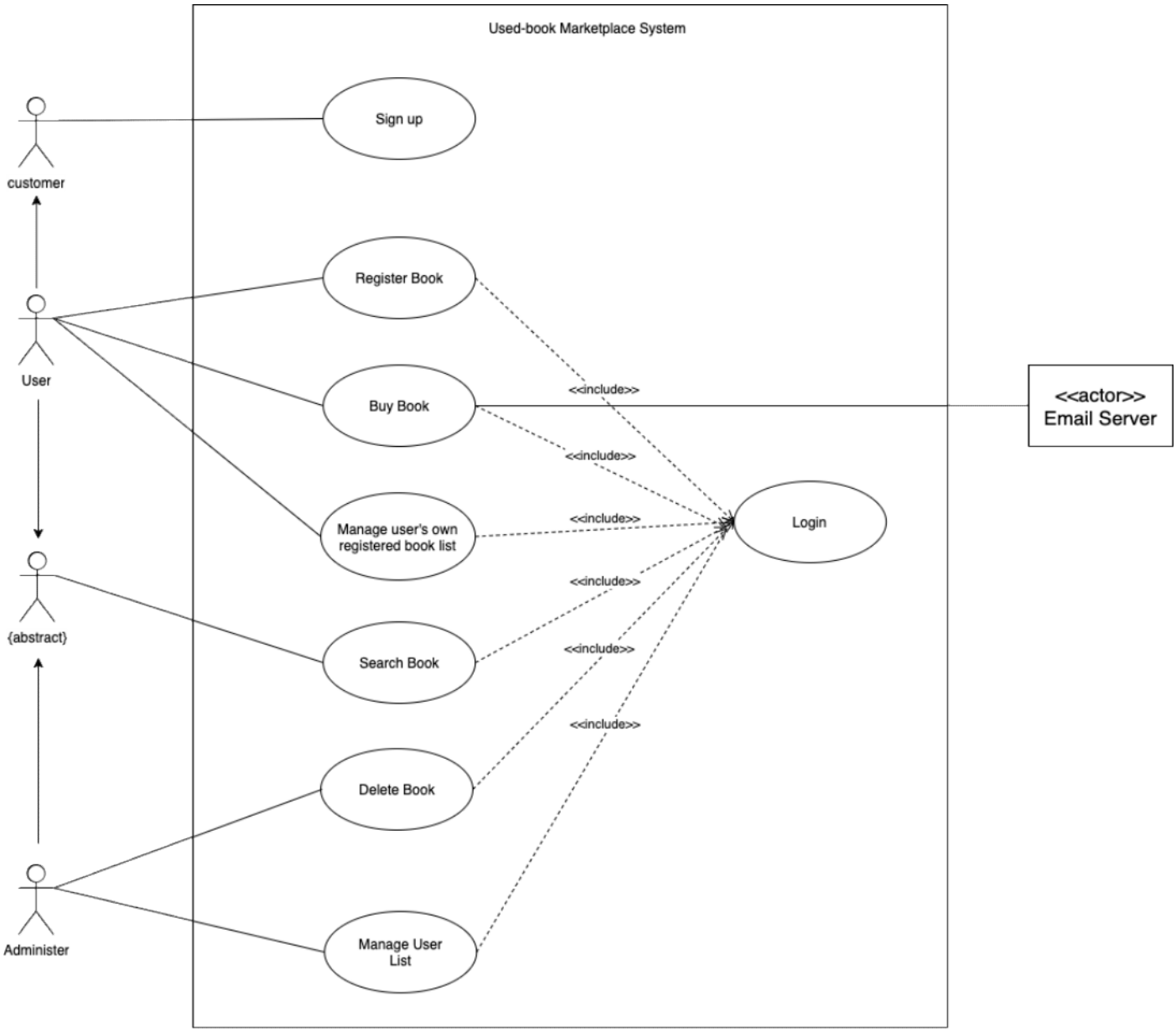
SE 2020 봄학기 텀프로젝트

20170217 컴퓨터공학과 이승연

■ 목차

Ⅰ. 요구 정의 및 분석

1. Use Case Diagram



2. Use Case 명세

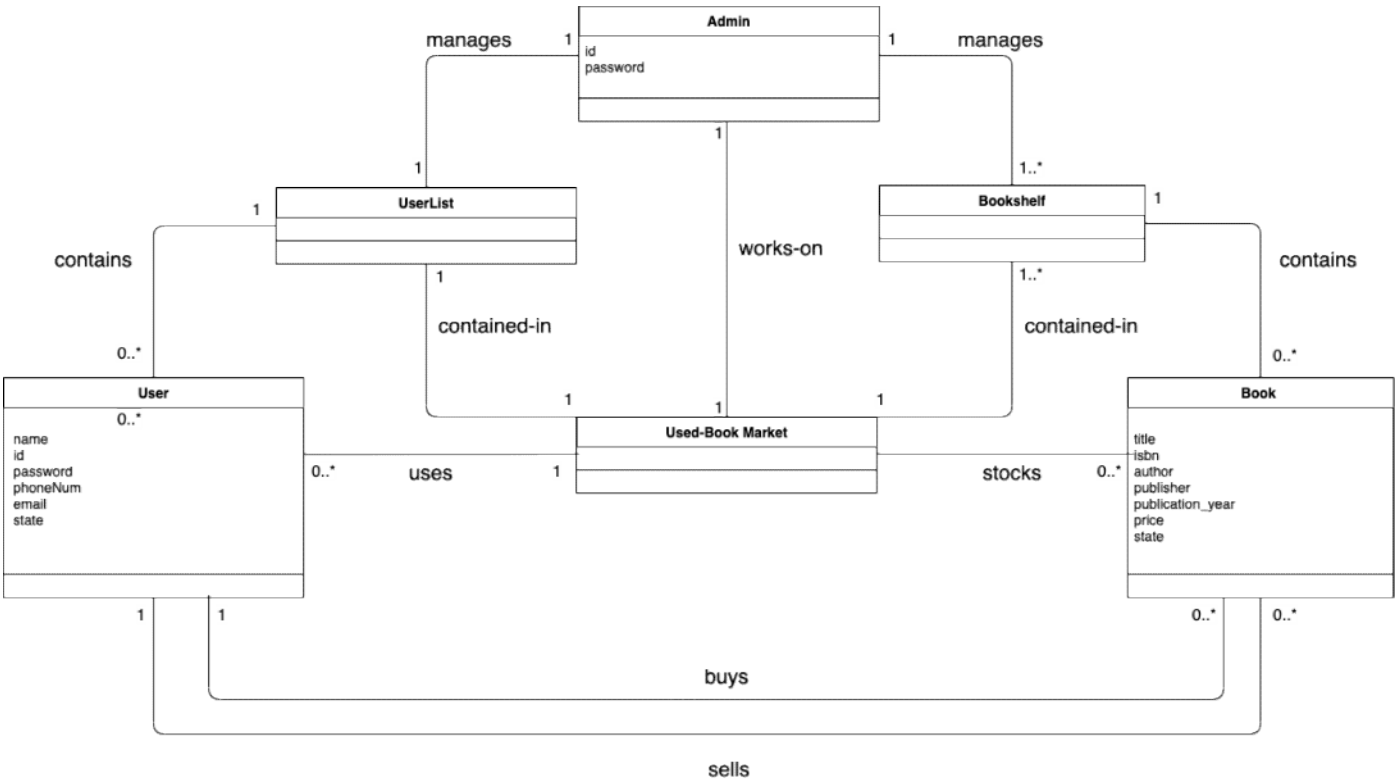
유스케이스 이름	Buy Book
참여 액터들	User
요약 설명	시스템의 사용자가 시스템에 등록된 책을 산다.
선행 조건	사용자가 이 시스템에 가입이 되어있고, 로그인된 상태여야 한다. (activated한 상태)
기본 흐름	1. 사용자가 로그인 하면, 시스템은 사용자가 이용할 수 있는 기능(책 검색하기, 책 구입하기, 책 등록하기, 본인이 등록한 책 목록보기(삭제 및 수정하기))을 화면에 보여주고, 사용자는 "책 구입하기"를 선택한다. 2. 사용자는 제목, ISBN 번호, 저자, 출판사, 출판년도, 판매자id별로 책을 검색할 수 있다. 그 중 선택하여 자신이 구매하고자 하는 책의 조건을 검색한다. 3. 시스템은 사용자의 구매 조건에 맞는 책의 목록을 보여준다. 4. 사용자는 그 중 자신이 사고자 하는 책을 골라 구매 요청한다. 5. 시스템은 사용자가 구매 요청한 책에 구매 표시하고, 구매 의사를 구매자와 판매자의 이메일로 발송한다. (이메일은 외부의 서비스를 사용하는 것으로 가정하고 실제 구현은 화면에 구매자와 판매자의 이메일을 표시하고 그 메일 주소로 메일이 발송되었다고만 출력한다.)
대체 흐름	1. 기본흐름 1번에서('책 구매하기' 선택 후) 등록된 책이 없으면 기본 흐름 2번으로 가지 못하고, "등록되어있는 책이 없습니다"라는 메세지를 띄운 후 책 구매가 종료된다. 2. 기본흐름 2번에서 사용자가 구매하고자 하는 조건에 맞는 책이 존재하지 않는다. 그렇다면 기본 흐름 3번으로 가지 못하고, "조건에 맞는 책이 없습니다"라는 메시지를 띄운 후 책 구매가 종료된다. 3. 기본흐름 3번에서 책의 목록을 보여줬는데, 그 중 사용자가 구매하고자 하는 책이 없으면 구매하지 않고 책 구매가 종료된다. 4. 기본흐름 4번에서 사용자가 사고자하는 책의 등록자인 경우, 회수 여부를 물은 후, 책 구매가 종료된다.
종료 조건	구입 의사를 나타내는 이메일이 정상적으로 보내져야 한다.

유스케이스 이름	Register Book
참여 액터들	User
요약 설명	사용자가 시스템에 자신이 등록한 책을 삭제하거나 수정한다.
선행 조건	사용자가 이 시스템에 가입이 되어있고, 로그인된 상태여야 한다. (activated한 상태)
기본 흐름	1. 사용자가 로그인 하면, 시스템은 사용자가 이용할 수 있는 기능(책 검색하기, 책 구입하기, 책 등록하기, 본인이 등록한 책 목록보기(삭제 및 수정하기))을 화면에 보여주고, 사용자는 "책 등록하기"를 선택한다. 2. 사용자는 팔고자하는 책을 등록하기 위해서 시스템에서 요구하는 사항들(책의 제목, 출판사, 저자정보, 출판년도, 가격, 책의 상태)을 적어 등록한다.(제목은 필수기입, 다른 건 선택사항, 책의 판매자아아디는 로그인된 아이디로 자동기입된다.) 이는 시스템에 저장된다. 3. 시스템은 "책을 등록하였습니다."라는 메시지를 띄운다.
대체 흐름	1. 기본 흐름 2번에서 사용자가 책의 제목을 기입하지 않았을 경우, "제목을 적지 않았습니다. 다시 입력해주세요." 라는 메시지를 띄운 후 기본 흐름 2번으로 돌아가 책 정보를 다시 기입하도록 한다.
종료 조건	사용자가 입력한 책의 정보가 정상적으로 추가돼야 한다.

유스케이스 이름	Delete Book
참여 액터들	Administer
요약 설명	관리자가 분쟁 등이 발생한 경우 문제의 책을 시스템에서 검색하여 삭제할 수 있다.
선행 조건	관리자가 올바른 아이디와 비밀번호를 통해 로그인된 상태여야 한다.
기본 흐름	1. 관리자가 로그인하면, 시스템은 관리자가 이용할 수 있는 기능(책 검색하기, 책 삭제하기, 전체 사용자 목록보기, 사용자 상태 전환하기, 사용자 삭제하기)을 화면에 보여주고, 관리자는 "책 삭제하기"를 선택한다. 2. 관리자는 제목, ISBN 번호, 저자, 출판사, 출판년도, 판매자id별로 책을 검색할 수 있다. 그 중 검색조건을 선택하여 문제가 발생한 책의 조건을 검색한다. 3. 시스템은 그 조건에 맞는 책들의 목록을 보여준다. 4. 관리자는 그 중 문제가 된 책을 선택하여 삭제한다. 5. 시스템은 "책이 삭제되었습니다." 라는 메세지를 띄운다.
대체 흐름	1. 기본흐름1번에서('책 삭제하기' 선택 후) 등록된 책이 없으면 기본 흐름 2번으로 가지 못하고, "등록되어있는 책이 없습니다"라는 메세지를 띄운 후 책 구매가 종료된다. 2. 기본흐름 2번에서 관리자가 찾는 조건에 맞는 책이 존재하지 않는다면 기본 흐름 3번으로 가지 못하고, "조건에 맞는 책이 없습니다"라는 메시지를 띄운다. 3. 기본흐름 4번에서 조건에 맞는 책들 중 관리자가 삭제하고자 하는 책이 없으면 삭제하지 않고 책 삭제하기가 종료된다.
종료 조건	관리자가 삭제한 책이 시스템에서 정상적으로 삭제돼야 한다.

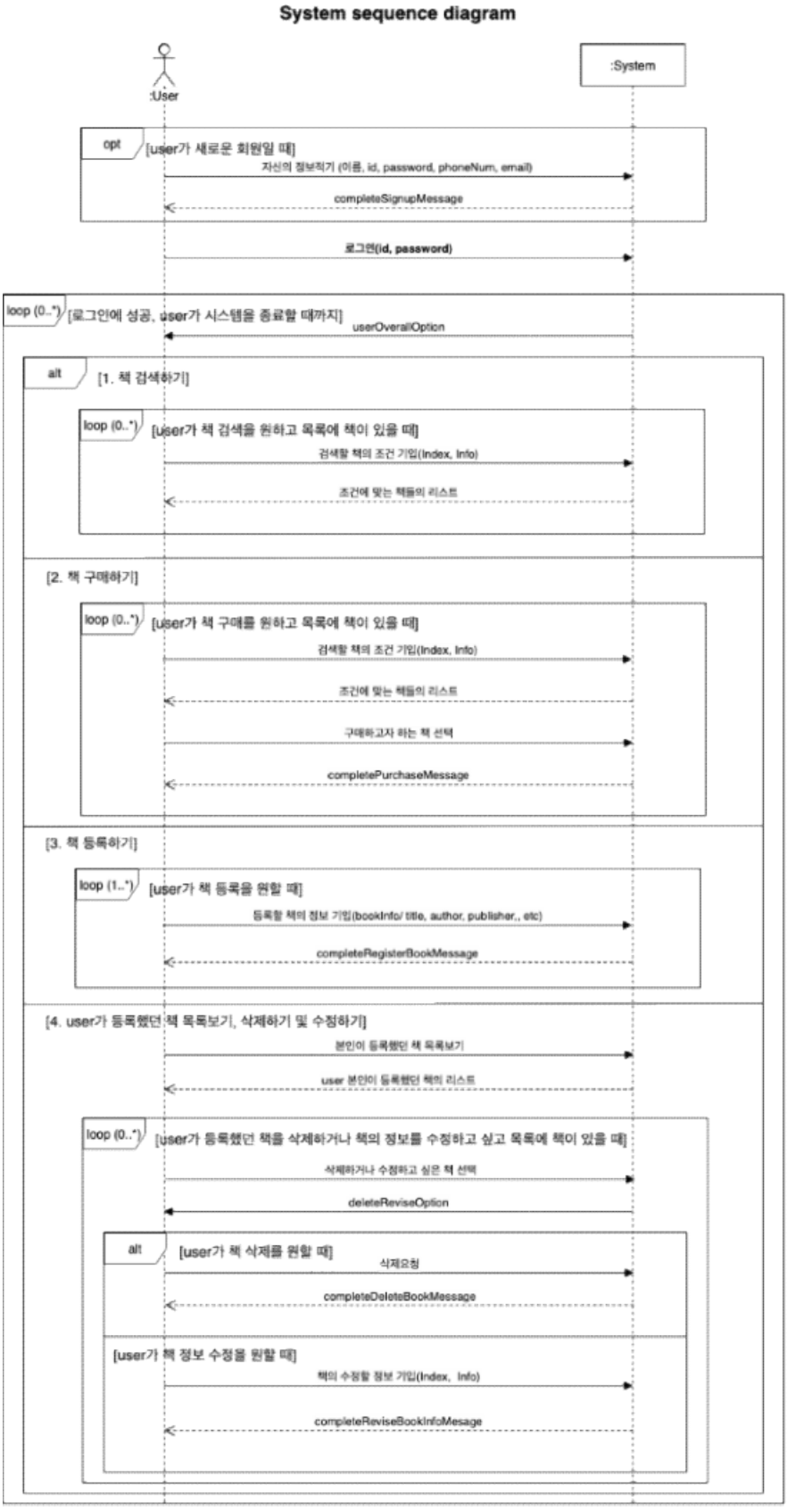
유스케이스 이름	Convert State of User
참여 액터들	Administer
요약 설명	관리자가 사용자를 선택하여 activated 혹은 deactivated 상태로 전환한다.
선행 조건	관리자가 올바른 아이디와 비밀번호를 통해 로그인된 상태여야 한다.
기본 흐름	1. 관리자가 로그인하면, 시스템은 관리자가 이용할수 있는 기능(책 검색하기, 책 삭제하기, 전체 사용자 목록보기, 사용자 상태 전환하기, 사용자 삭제하기)을 화면에 보여주고, 관리자는 "사용자 상태 전환하기"를 선택한다. 2. 관리자는 사용자 리스트에 있는 사용자들 중 상태를 전환하고 싶은 사용자를 선택하여 상태를 바꾸어준다. (activated->deactivated / deactivated->activated) 3. 시스템은 "사용자의 상태가 전환됐습니다." 라는 메세지를 띄운다.
대체 흐름	1. 기본 흐름 1번에서('사용자 상태 전환하기' 선택 후) 시스템이 등록되어있는 사용자가 존재하지 않는다면, "등록되어있는 사용자가 없습니다"라는 메세지를 띄운 후 사용자 상태 변경하기가 종료된다.
종료 조건	관리자가 상태가 전환된 사용자의 정보가 시스템에서 정상적으로 전환돼야 한다.

3. Domain Model

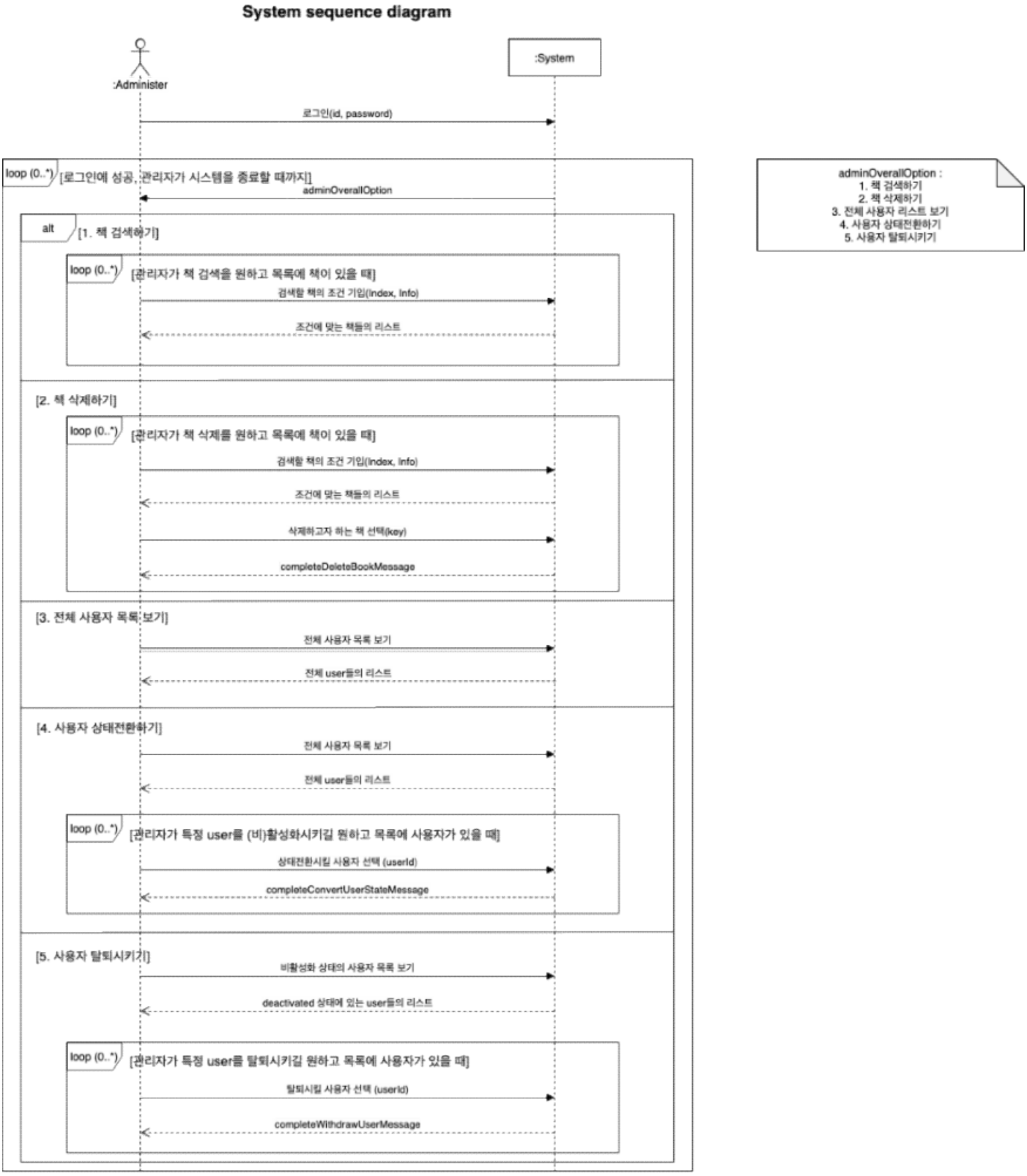


4. System Sequence Diagram

1) Actor가 User일 때



2) Actor가 Administer일 때



5. Operation Contract

Operation	검색할 책의 조건 기입(index : String, info : String)
Cross References	Use case : Buy Book
Preconditions	- 사용자가 로그인이 되고, '책 구매하기'를 선택. - 목록에 책이 있음.
Postconditions	- 조건에 맞는 책을 찾아 리스트에 저장하고 이를 보여줌. (instance creation, attribute modification) - 책의 목록에서 조건에 맞는 책을 찾으므로 책 목록과 관계가 생김. (association formed)

Operation	등록할 책의 정보 기입(title : String, ISBN : String, author : String, publisher : String, publication year : String, state: String)
Cross References	Use case : Register Book
Preconditions	- 사용자가 로그인이 되고, '책 등록하기'를 선택.
Postconditions	- 입력한 조건의 book instance가 생성됨. (instance creation) - 책의 목록에 저장하므로 책 목록과의 관계가 생김. (association formed)

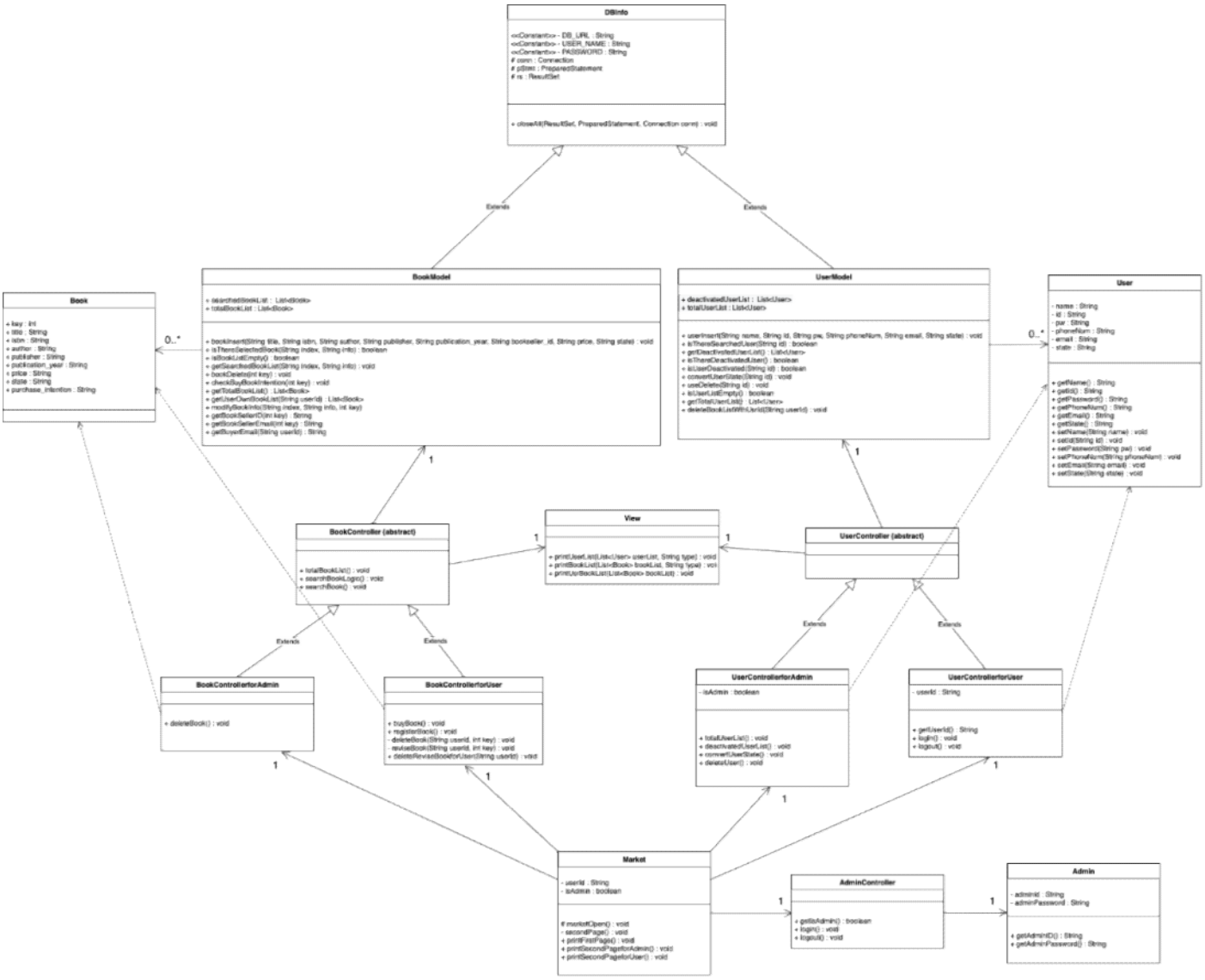
Operation	상태전환시킬 사용자 선택(userId : String)
Cross References	Use case : Convert State of User
Preconditions	- 관리자가 로그인이 되고, '사용자 상태전환하기'를 선택.
Postconditions	- 선택한 사용자의 상태가 전환됨. (attribute modification) - 선택한 사용자의 바뀐 상태가 저장되므로 사용자 목록과의 관계가 생김. (association formed)

Ⅱ. 설계

- UI와 응용 Logic의 구분

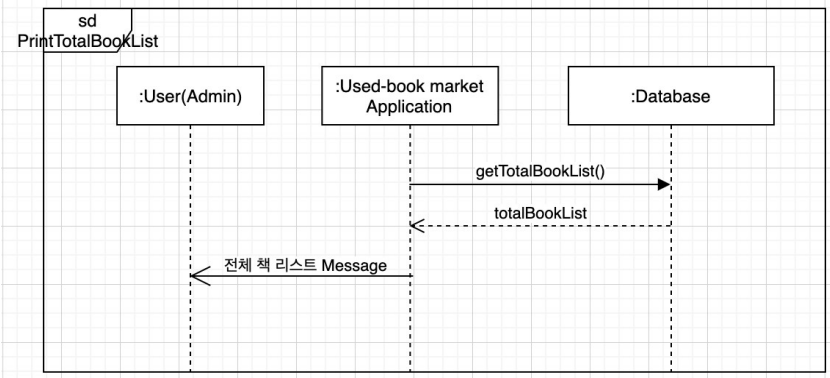
여러 software architecture 스타일 중 MVC 패턴을 적용하여 시스템을 구현하고자 하였다. 즉, 모델에서는 데이터와 데이터를 처리하는 핵심기능을 포함하며, 뷰는 사용자에게 정보를 표시하고, 컨트롤러는 사용자의 입력을 처리하여 모델이 어떠한 기능을 수행해야 하는지 알려주는 역할을 한다. 이에 대한 자세한 구현내용은 'Ⅲ-2. 구현에 대한 설명 및 소스코드'에서 다룬다.

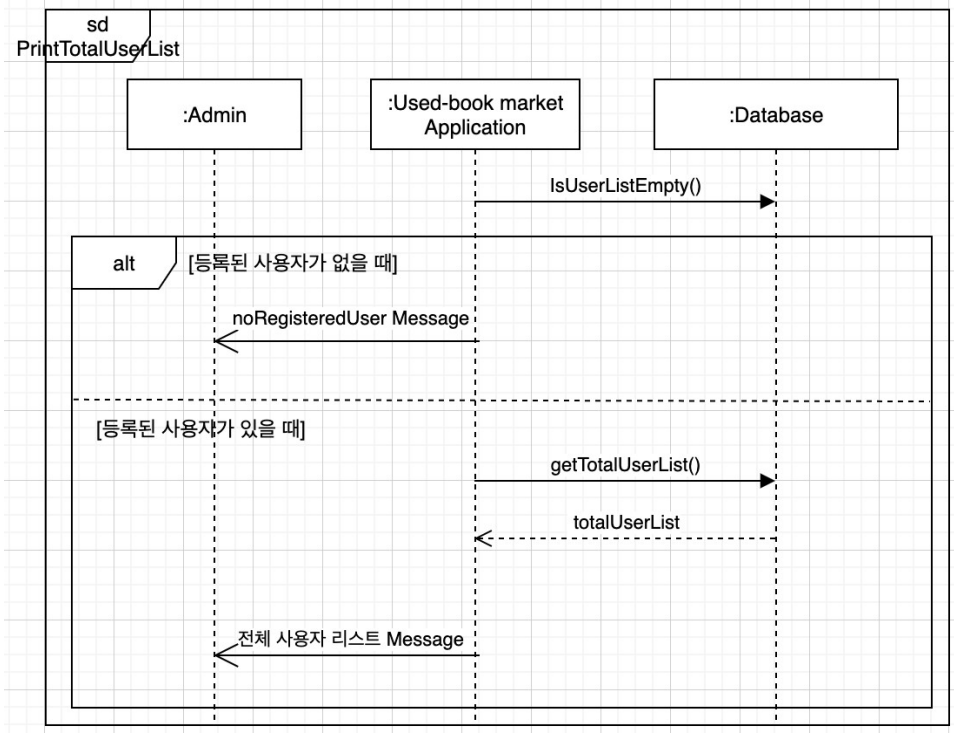
1. Design Class Diagram



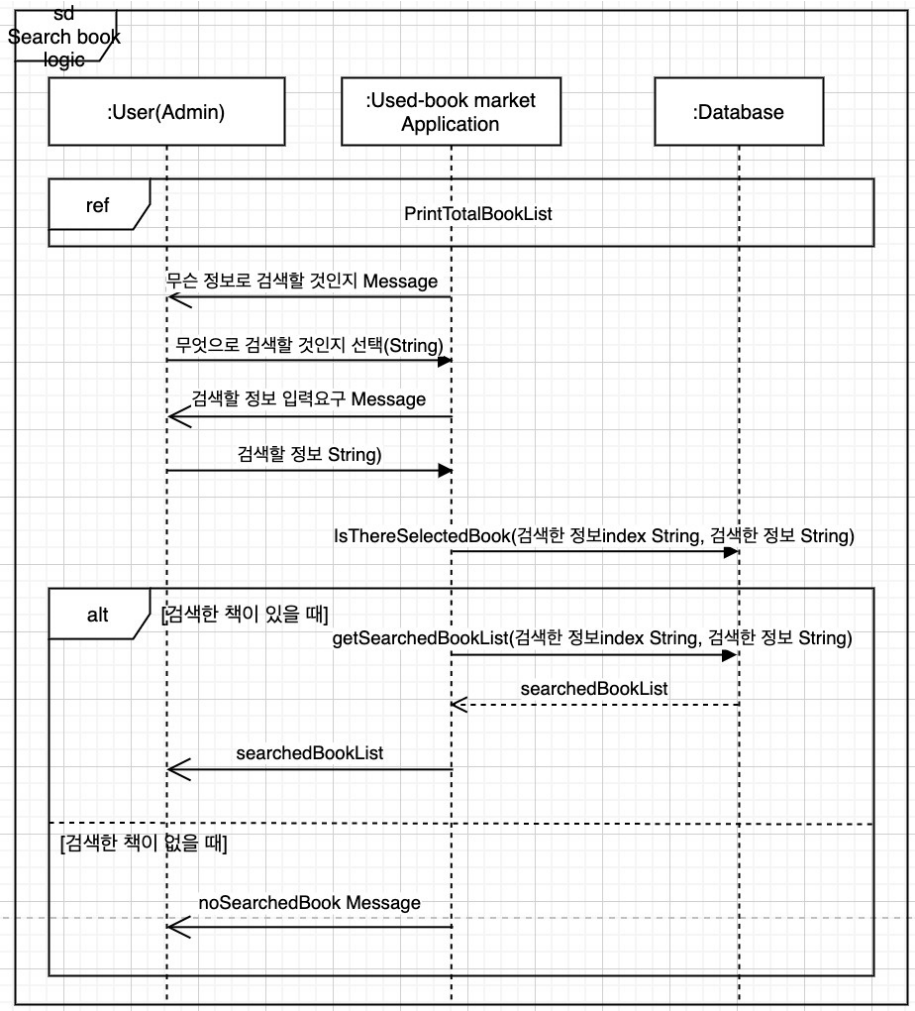
2. Sequence Diagram

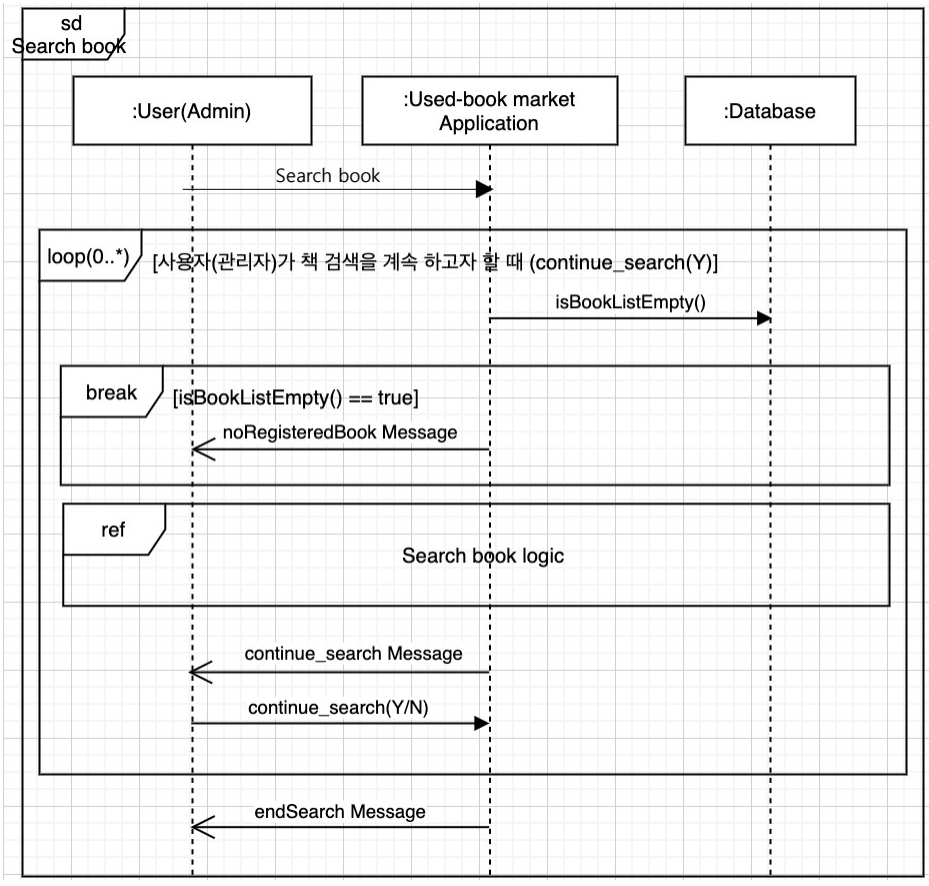
- 로그인 및 공통적으로 쓰이는 print함수 (아래의 sequence diagram에서 쓰임)



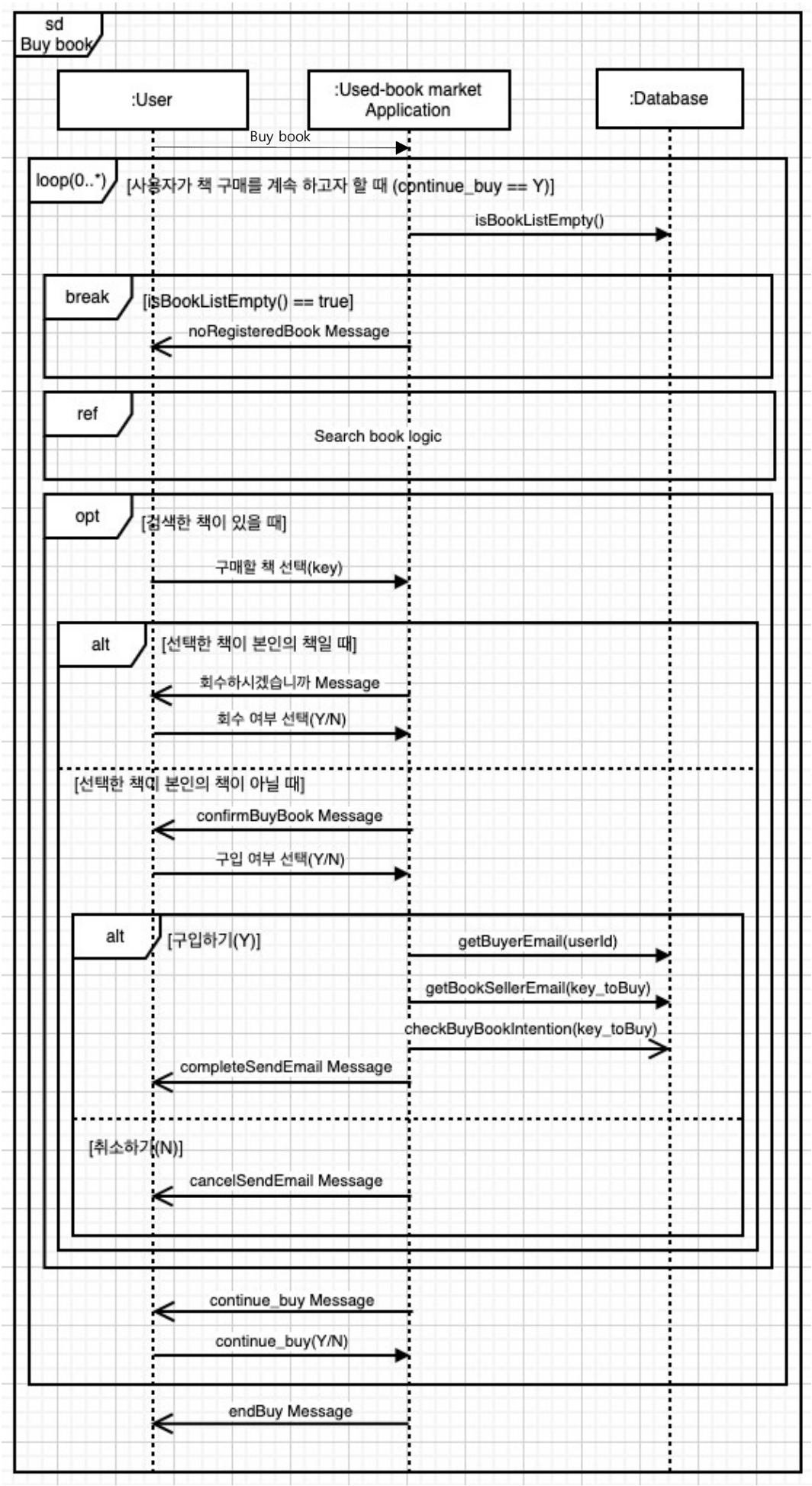


- 책 검색하기

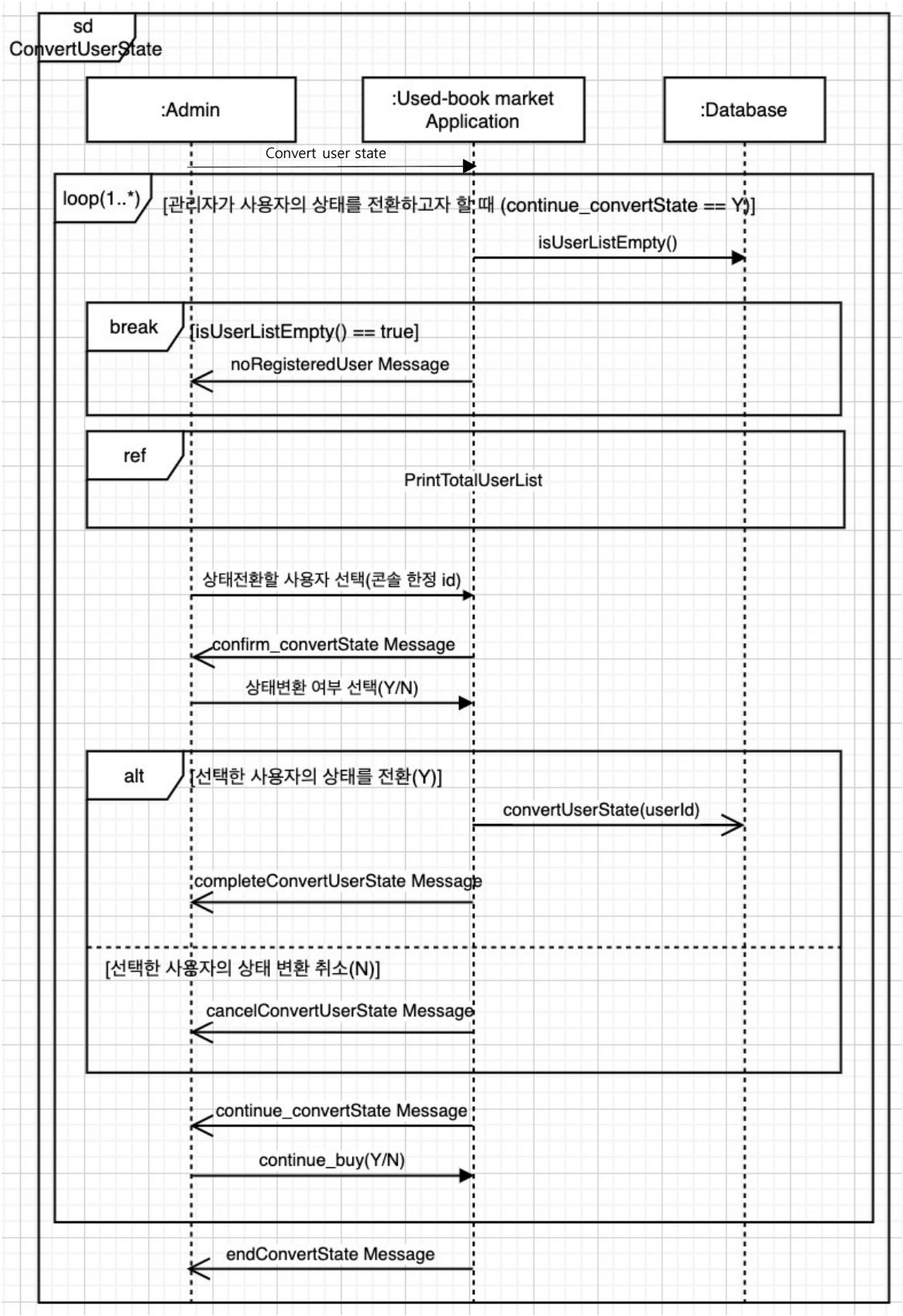

sd
Search book logic
:User(Admin)
:Used-book market Application
:Database
ref
PrintTotalBookList
무슨 정보로 검색할 것인지 Message
무엇으로 검색할 것인지 선택(String)
검색할 정보 입력요구 Message
검색할 정보 String)
IsThereSelectedBook(검색한 정보index String, 검색한 정보 String)
alt
[검색한 책이 있을 때]
getSearchedBookList(검색한 정보index String, 검색한 정보 String)
searchedBookList
searchedBookList
[검색한 책이 없을 때]
noSearchedBook Message


sd
Search book
:User(Admin)
:Used-book market Application
:Database
Search book
loop(0..*)
[사용자(관리자)가 책 검색을 계속 하고자 할 때 (continue_search(Y)]
isBookListEmpty()
break
[isBookListEmpty() == true]
noRegisteredBook Message
ref
Search book logic
continue_search Message
continue_search(Y/N)
endSearch Message

- 책 구매하기


sd
Buy book
:User
:Used-book market Application
:Database
Buy book
loop(0..*)
[사용자가 책 구매를 계속 하고자 할 때 (cpntinue_buy == Y)]
isBookListEmpty()
break
[isBookListEmpty() == true]
noRegisteredBook Message
ref
Search book logic
opt
[검색한 책이 있을 때]
구매할 책 선택(key)
alt
[선택한 책이 본인의 책일 때]
회수하시겠습니까 Message
회수 여부 선택(Y/N)
[선택한 책이 본인의 책이 아닐 때]
confirmBuyBook Message
구입 여부 선택(Y/N)
alt
[구입하기(Y)]
getBuyerEmail(userId)
getBookSellerEmail(key_toBuy)
checkBuyBookIntention(key_toBuy)
completeSendEmail Message
[취소하기(N)]
cancelSendEmail Message
continue_buy Message
continue_buy(Y/N)
endBuy Message

- 사용자 상태 전환하기


sd
ConvertUserState
:Admin
:Used-book market Application
:Database
Convert user state
loop(1..*)
[관리자가 사용자의 상태를 전환하고자 할 때 (continue_convertState == Y)]
isUserListEmpty()
break
[isUserListEmpty() == true]
noRegisteredUser Message
ref
PrintTotalUserList
상태전환할 사용자 선택(콘솔 한정 id)
confirm_convertState Message
상태변환 여부 선택(Y/N)
alt
[선택한 사용자의 상태를 전환(Y)]
convertUserState(userId)
completeConvertUserState Message
[선택한 사용자의 상태 변환 취소(N)]
cancelConvertUserState Message
continue_convertState Message
continue_buy(Y/N)
endConvertState Message

Ⅲ. 구현 및 테스트

1. 내가 더 나은 설계 및 구현을 위해 노력한 점 (EXTRA POINT)

● 데이터베이스의 사용 (MySQL)

처음 시작했을 때는 배열리스트를 이용하여 구현을 하였으나, 데이터베이스를 이용하면 배열리스트보다 확장성이 있을 거라 생각하여, 데이터베이스로 설계를 변경하여 구현하였다.

배열리스트와 파일시스템을 이용한 프로그램은 local PC에서 여러 명이 사용하면서 그때마다 업데이트를 할 수 없지만, 데이터베이스를 이용할 경우 서버와 클라이언트로 분리하여 동시에 여러 명이 이용할 수도 있고, 웹서버와 브라우저를 기반으로 하는 웹서비스로 변환하는 것도 가능하여 확장성이 좋다고 판단하였다.

이론적으로 공부해본 것 외에는 DB 설계와 구현은 처음이라 에러 처리할 것이 많아 힘들고 어려운 과정이었지만, 결과적으로 데이터베이스를 이용한 결과가 잘 나와서 뿌듯하다.

```
ublic class BookList extends ArrayList<Book>{
   Scanner scnr = new Scanner(System.in);

   public void addBook(String title, int isbn, String author, String publisher, int publication_year, String bookseller_id, int price, String state)
   {
       Book book = new Book();

       book.setTitle(title);
       book.setISBN(isbn);
       book.setAuthor(author);
       book.setPublisher(publisher);
       book.setPublicationYear(publication_year);
       book.setBookSellerID(bookseller_id);
       book.setPrice(price);
       book.setState(state);

       this.add(book);
   }
```

▶배열리스트로 짜던 코드의 일부

```
public void insertBook(Book book) {
    String query = "INSERT INTO book VALUE (null, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    try {
        conn = DriverManager.getConnection(DB_URL,USER_NAME,PASSWORD);
        pStmt = conn.prepareStatement(query);
        pStmt.setString(1, book.getTitle());
        pStmt.setString(2, book.getISBN());
        pStmt.setString(3, book.getAuthor());
        pStmt.setString(4, book.getPublisher());
        pStmt.setString(5, book.getPublicationYear());
        pStmt.setString(6, book.getBookSellerID());
        pStmt.setString(7, book.getPrice());
        pStmt.setString(8, book.getState());
        pStmt.setString(9, book.getPurchaseIntention());
        pStmt.executeUpdate();
    }
    catch(Exception e) {
        e.printStackTrace();
    }
    finally {
        closeAll(rs, pStmt, conn);
    }
```

▶데이터베이스를 이용한 현재 프로그램의 일부

● MVC pattern 적용

Ⅱ 설계 부분에서 조금 다루었듯이 MVC pattern을 적용함으로써 유지보수성과 단위테스트에 용이한 설계를 하였다. 조금 더 자세한 관련내용은 아래의 **2. 구현에 대한 설명 및 소스코드**에서 다룬다.

● 주어진 조건보다 서비스를 조금 더 편하게 만듦

1) 회원가입 후 로그인이 바로 가능하다. 또한, 로그인 가능 횟수는 세 번으로 제한했다.

2) 사용자의 '책 구매' 기능에서, 자신이 등록했던 책을 선택하여 구매하고자 한다면, '본인이 등록한 책입니다. 회수하시겠습니까?'라는 메시지와 함께 회수가 가능하도록 만들었다.

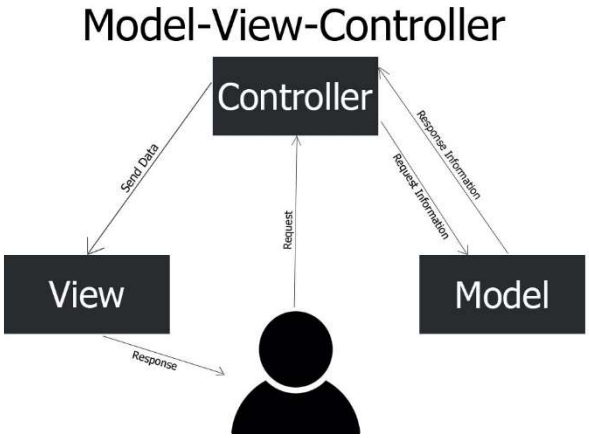
3) 조건에서 이 시스템의 '책 구매' 기능은 실제로 시스템에서 구매가 이루어지는 것이 아닌, 구매의사만 나타내고 실제 판매자와 구매자가 만나 거래가 이루어진 후에 판매자가 직접 그 책을 목록에서 삭제하는 것이다. 이를 조금 더 편리하게 하기위해, 책을 구매하고자 하는 사용자가 구매의사를 나타내면 그 책이 구매의사가 표시된 책임을 나타내도록 했다.

4) 현실에서 이러한 프로그램을 사용한다고 가정했을 때, 한 기능을 계속해서 사용하는 경우가 많을 것 같아, 사용자나 관리자가 책 구매, 책 등록, 책 삭제 등 프로그램의 기능들을 이용할 때에 계속해서 사용하고 싶다면, 처음으로 돌아가지 않고 계속 그 기능을 이용할 수 있도록 하였다.

2. 구현에 대한 설명 및 소스코드

● MVC pattern 적용

UI와 응용 Logic을 분리하여 추후 UI가 바뀌더라도 로직 이하의 계층은 변경없이 재사용할 수 있도록 하기 위해 MVC 개념을 최대한 적용하여 시스템을 구현하였다. 단, 콘솔로 구현을 하였기 때문에, controller와 view를 완전하게 분리하기는 (한줄짜리 print함수가 여러 개 존재하는 등) 코드가 너무 복잡해져, non-UI인 model과 나머지 view, controller의 구분만을 확실히 하였다. 즉, 실제 콘솔에 출력이 나타나고 사용자의 입력을 받아 model이



어떤 일을 수행해야하는지 전달해주는 부분과, 그 전달을 받아 data와 관련해서 처리해야 할 일(함수)을 하는 부분인 model을 구분하였다. 또한 model에서 가져온 리스트를 출력하는 부분만 view쪽에 구현하였다

▶ 이번 프로젝트에서 콘솔로 구현하기 위해 생각한 MVC패턴이다.

(출처: https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66)

구현한 코드 중 이것을 가장 간단하게 설명할 수 있는 부분은 다음과 같다.

Controller부분

```
//책 검색
public void searchBookMain() {
   totalBookList();

   System.out.print("어떤 정보로 찾으시겠습니까? >> ");
   String index = scnr.next();
   while((index.equals("제목") || index.equals("ISBN") || index.equals("저자") || index.equals("출판사") || index.equals("출판년도") || index.equals("판매자id")) == false){
       System.out.print("잘못 입력하셨습니다. 다시 입력해주세요. >> ");
       index = sn.nextLine();
   }
   System.out.printf("책의 %s를 입력해주세요.>> ", index);
   String info = sn.nextLine();
   System.out.println("");
   if(bookModel.isThereSelectedBook(index, info)) {
       bookModel.setSearchedBookList(index, info);
       printBook.printBookList(bookModel.searchedBookList, "searched");
   }
   else {
       bookModel.setSearchedBookList(index, info);
       System.out.println("조건을 만족하는 책이 존재하지 않습니다.");
       System.out.println("");
   }
}
```

▶ 이렇게, 사용자의 입력을 받아오는 부분은 controller쪽에 구현하였다.

```
//전체 책 목록 보기
public void totalBookList(){
    bookModel.setTotalBookList();
    printBook.printBookList(bookModel.totalBookList, "total");
}
```

▶ controller는 model로부터 전체 목록을 받아와 view에 넘겨준다.

Model부분

```
//전체 책 목록
public void setTotalBookList(){
    totalBookList.clear();
    String query = "SELECT * FROM book";
    try {
        conn = DriverManager.getConnection(DB_URL,USER_NAME,PASSWORD);
        pStmt = conn.prepareStatement(query);
        rs = pStmt.executeQuery(query);
        while(rs.next()) {
            Book book = new Book();
            book.setKey(rs.getInt("uniquekey"));
            book.setTitle(rs.getString("title"));
            book.setISBN(rs.getString("isbn"));
            book.setAuthor(rs.getString("author"));
            book.setPublisher(rs.getString("publisher"));
            book.setPublicationYear(rs.getString("publication_year"));
            book.setBookSellerID(rs.getString("bookseller_id"));
            book.setPrice(rs.getString("price"));
            book.setState(rs.getString("state"));
            book.setPurchaseIntention(rs.getString("purchase_intention"));

            totalBookList.add(book);
        }
    }
    catch(Exception e){
        e.printStackTrace();
    }
    finally {
        closeAll(rs, pStmt, conn);
    }
}
```

▶ model은 데이터베이스로부터 query해서 데이터를 가져와 처리한다.

View부분

```
public void printBookList(List<Book> bookList, String type){
    switch(type) {
    case "total":
        System.out.println("<<전체 책 목록>>");
        break;

    case "searched":
        System.out.println("<<검색된 책 목록>>");
        break;
    }

    for(Book book : bookList) {
        System.out.printf("%d|제목: %-20.20s |ISBN: %-10.10s |저자: %-15.15s |출판사: %-10.10s |출판년도: %-10.10s |판매자id: %-20.20s |가격: %-15.15s |상태: %-15.15s |구매의도: %-5.5s\n",
                book.getKey(), book.getTitle(), book.getISBN(), book.getAuthor(), book.getPublisher(), book.getPublicationYear(), book.getBookSellerID(), book.getPrice(), book.getState(), book.getPurchaseIntention()
    }
    System.out.println("");
}
```

▶ view는 controller로부터 받은 리스트를 띄운다.

● 콘솔 인터페이스로 구현함에 따라 추가한 예외처리

- 콘솔로 사용자의 입력을 받아 처리하려면, 그 입력이 유일한 값이어야 한다. 예를 들어, 책을 구매할 때에 보여준 목록에서 어떤 책을 구매하고 싶은지 결정을 하기 위해서는 유일한 값이 존재해야한다. 사용자의 attribute중에는 사용자id가 unique한 값으로 존재한다. (회원가입 시에 현재 사용자 목록에 있는 사용자들의 id와 같은 id를 쓰지 못하도록 함.) 하지만 주어진 조건에서 책에는 unique한 값이 없었다. 따라서 unique한 값인 key를 책 class의 attribute에 추가하였고 데이터베이스에는 auto increment한 값으로 설정하였다.

- 또한 콘솔로 사용자의 입력을 받을 때 발생하는 예외들이 많은데, 이를 처리하였다.

```
//key 값에 int가 아닌 다른 유형의 값을 넣었을 때의 예외처리(콘솔한정)
while(true) {
    try{
        key_toDelete = scnr.nextInt();
        break;
    }
    catch(InputMismatchException e) {
        scnr = new Scanner(System.in);
        System.out.print("잘못 입력하셨습니다. 삭제할 책의 번호를 다시 입력해주세요.(책 삭제 종료하기 0) >> ");
    }
}
```

```
//콘솔한정
while((continue_deleteBook.equals("Y") || continue_deleteBook.equals("N")) == false) {
    System.out.print("잘못입력하셨습니다. 다시 입력해주세요.[Y/N] >> ");
    continue_deleteBook = scnr.nextLine();
}
```

```
//콘솔한정
do{
    //삭제 진행
    if(key_toDelete != 0) {
        //입력한 번호의 책이 찾은 책목록 중 존재하는지 확인(콘솔한정)
        for(Book book: bookDAO.searchedBookList) {
            if(book.key == key_toDelete) {
                bookWthKey = true;
                break;
            }
        }
        //입력한 번호의 책이 찾은 책목록 중 없을 때
        if(bookWthKey == false) {
            System.out.print("입력한 번호의 책이 없습니다. 다시 입력해주세요. >> ");
            //key 값에 int가 아닌 다른 유형의 값을 넣었을 때의 예외처리(콘솔한정)
            while(true) {
                try{
                    key_toDelete = scnr.nextInt();
                    break;
                }
                catch(InputMismatchException e) {
                    scnr = new Scanner(System.in);
                    System.out.print("잘못 입력하셨습니다. 책의 번호를 다시 입력해주세요. >> ");
                }
            }
        }
    }
    //삭제 종료하기(0)
    else
        break loop;
} while(bookWthKey == false);
```

● 전체적인 코드 구조

- **Model**

: book, user, admin의 각각의 클래스와 book, user 데이터를 다루는 bookModel, userModel이 있다. 데이터를 데이터베이스로 관리하기 때문에 클래스의 대부분은 쿼리하는 함수로 되어있다.

```
public class Book {
    //book의 정보 : 제목, ISBN, 저자, 출판사, 출판년도, 판매자id, 가격, 상태(Excellent, Good, Fair)
        int key = 0; //book에는 unique한 값이 없어서 추가, db에서 ai
        private String title;
        private String isbn;
        private String author;
        private String publisher;
        private String publication_year;
        private String bookseller_id;
        private String price;
        private String state;
        private String purchase_intention = "N";

        public Book() {}

        public Book(String title, String isbn, String author, String publisher,
                String publication_year, String bookseller_id, String price, String state)
        {
            this.title = title;
            this.isbn = isbn;
            this.author = author;
            this.publisher = publisher;
            this.publication_year = publication_year;
            this.bookseller_id = bookseller_id;
            this.price = price;
            this.state = state;
        }

        //getter
        public int getKey() {return key;}
        public String getTitle() {return title;}
        public String getISBN() {return isbn;}
        public String getAuthor() {return author;}
        public String getPublisher() {return publisher;}
        public String getPublicationYear() {return publication_year;}
        public String getBookSellerID() {return bookseller_id;}
        public String getPrice() {return price;}
        public String getState() {return state;}
        public String getPurchaseIntention() {return purchase_intention;}

        //setter
        public void setKey(int key) {this.key = key;}
        public void setTitle(String title) {this.title = title;}
        public void setISBN(String isbn) {this.isbn = isbn;}
        public void setAuthor(String author) {this.author = author;}
        public void setPublisher(String publisher) {this.publisher = publisher;}
        public void setPublicationYear(String publication_year) {this.publication_year = publication_year;}
        public void setBookSellerID(String bookseller_id) {this.bookseller_id = bookseller_id;}
        public void setPrice(String price) {this.price = price;}
        public void setState(String state) {this.state = state;}
        public void setPurchaseIntention(String purchase_intention) {this.purchase_intention = purchase_intention;}
}
```

```
public class User {
    //user의 정보 : 이름, id, 비밀번호, 전화번호, 이메일, 상태(activated, deactivated)
    private String name;
    private String id;
    private String pw;
    private String phoneNum;
    private String email;
    private String state = "activated";

    public User(){}

    public User(String name, String id, String pw, String phoneNum, String email)

    //getter
    public String getName() {return name;}
    public String getId() {return id;}
    public String getPassword() {return pw;}
    public String getPhoneNum() {return phoneNum;}
    public String getEmail() {return email;}
    public String getState() {return state;}

    //setter
    public void setName(String name) {this.name = name;}
    public void setId(String id) {this.id = id;}
    public void setPassword(String pw) {this.pw = pw;}
    public void setPhoneNum(String phoneNum) {this.phoneNum = phoneNum;}
    public void setEmail(String email) {this.email = email;}
    public void setState(String state) {this.state = state;}
}
```

```
public class Admin  {
    private String adminId = "admin";
    private String adminPassword = "nayana";

    public String getAdminID() {return adminId;}
    public String getAdminPassword() {return adminPassword;}
}
```

```java
public class BookModel extends DBInfo{
    //사용자와 관리자가 찾는 책의 목록
    List<Book> searchedBookList = new ArrayList<Book>();
    List<Book> totalBookList = new ArrayList<Book>();

    //전체 책 목록
    public void setTotalBookList(){ ..

    //지금까지 등록된 책이 있나 확인
    public boolean isBookListEmpty() {..

    //사용자와 관리자가 찾는 책의 목록 set
    public void setSearchedBookList(String index, String info){..

    //주어진 제목, 저자, 판매자id의 정보를 가진 책이 존재하는지
    //사용자와 관리자가 책을 찾을 때 그 책이 있는지 없는지 확인(사용자, 관리자 1번 기능)
    public boolean isThereSearchedBook(String index, String info) {     ..

    //구매자(사용자)가 판매의도를 표현(사용자 2번 기능)
    public void checkBuyBookIntention(int key) {..

    //구매자(사용자)가 책을 구매시에(사용자 2번 기능)
    //판매자id
    //본인이 등록한 책 회수시에 본인확인
    public String getBookSellerID(int key) {..

    //판매자email
    public String getBookSellerEmail(int key) {..

    //구매자email
    public String getBuyerEmail(String userId) {..

    //사용자가 책 등록시(사용자 3번 기능)
    public void addBook(Book book) {..

    //사용자가 등록한 책이 있는지(사용자 4번 기능)
    public boolean isThereUserOwnBook(String userId) {..

    //사용자 본인이 등록한 책의 리스트, 사용자가 자신의 책 삭제 및 수정(사용자 4번 기능)
    public List<Book> getUserOwnBookList(String userId) {..

    //관리자가 문제가 되는 책 삭제(관리자 2번 기능), 사용자가 본인이 등록한 책 삭제(사용자 4번 기능)
    public void deleteBook(int key) {..

    //사용자가 본인이 등록한 책 수정할 때(사용자 4번 기능)
    public void modifyBookInfo(String index, String info, int key){..

    //junit test 때문에 추가
    public void truncateBookTable() {..

}
public class UserModel extends DBInfo{

    List<User> totalUserList = new ArrayList<User>();
    List<User> deactivatedUserList = new ArrayList<User>();


    //사용자 회원가입
    public void addUser(User user) {..

    //회원가입시에 동일한 id가 존재해서는 안됨(사용자 로그인)
    public boolean isThereSearchedUser(String id) {..

    //사용자가 입력한 아이디와 비밀번호를 가진 사용자가 사용자(회원가입된) 목록에 있는지(사용자 로그인)
    public boolean isThereUserWthIdPw(String id, String pw) {..

    //사용자가 비활성화 상태(로그인 불가상태)인지(사용자 로그인)
    public boolean isUserDeactivated(String id) {..

    //전체 목록 출력 (관리자 기능 3번, 4번)
    public void setTotalUserList(){..

    //사용자 상태 전환(관리자 기능 4번)
    public void convertUserState(String id) {..

    //비활성화된 사용자가 있는지(관리자 기능 5번)
    public boolean isThereDeactivatedUser() {..

    //비활성화 상태의 사용자 리스트(관리자 기능 5번)
    public void setDeactivatedUserList() {..

    //(관리자 기능 5번)
    public void deleteUser(String id) {..

    //등록되어있는 사용자가 없을 때 (관리자 기능 3번, 5번)
    public boolean isUserListEmpty() {..

    //사용자 탈퇴시 그 사용자가 등록했던 책들도 함께 삭제 (관리자 기능 5번)
    public void deleteBookListWthUsrId(String userId) {..

}
```

- **Controller**

: 크게보면 book에 관련된 기능을 관리하는 bookcontroller와, user에 관련된 기능을 관리하는 usercontroller와 admin에 관련된 기능을 관리하는 admincontroller가 존재한다. 또한 관리자와 사용자의 기능을 따로 관리하기 위해 각각 admin의 기능, user의 기능을 관리하는 부분을 하위클래스로 두었다. 관리자와 사용자의 공통적인 기능만 상위클래스인 bookcontroller와 usercontroller에서 관리한다.

```
public abstract class BookController{
    Scanner scnr = new Scanner(System.in);
    Scanner sn = new Scanner(System.in);
    BookModel bookModel = new BookModel(); //model객체 생성
    View view = new View(); //view객체 생성
    public BookController() {}

    //전체 책 목록 보기
    public void totalBookList(){..

    //책 검색
    public void searchBookMain() {..

    //책 검색 (관리자, 사용자 공통 1번 기능)
    public void searchBook() {..
}
```

```
public class BookControllerforAdmin extends BookController{
    public BookControllerforAdmin() {}

    //책 삭제 기능 (관리자 2번 기능)
    public void deleteBook(){   ..
}
```

```
public class BookControllerforUser extends BookController {
    Scanner scnr = new Scanner(System.in);
    Scanner scnrRegister = new Scanner(System.in);

    public BookControllerforUser() {}

    //책 구입 (사용자 2번 기능)
    public void buyBook(String userId) {..

    //책 등록 (사용자 3번 기능)
    public void registerBook(String bookSellerId) {..

    //책 삭제(deleteBook), 책 정보 수정(reviseBook)은 deleteReviseBookforUser를 위한 클래스 : private 선언
    //본인이 등록한 책 삭제 로직
    private void deleteBook(String userId, int key) {..

    //본인이 등록한 책 정보수정 로직
    private  void reviseBook(String userId, int key) {..

    //자기가 등록한 책 보여주고 삭제나 수정 선택 가능 (사용자 4번 기능)
    public void deleteReviseBookforUser(String userId) {..

}
```

```java
public abstract class UserController {
    UserModel userModel = new UserModel(); //model객체 생성
    View view = new View(); //view객체 생성

    public UserController() {}
}
```

```java
public class UserControllerforAdmin extends UserController{
    Scanner scnr = new Scanner(System.in);

    public UserControllerforAdmin() {}

    //전체 사용자 목록보기
    public void totalUserList(){..

    //비활성화 상태의 사용자 목록보기
    public void deactivatedUserList() { ..

    //사용자 상태 전환하기
    public void convertUserState() {..

    //사용자 탈퇴시키기
    public void withdrawUser() {..

}
```

```java
public class UserControllerforUser extends UserController{
    Scanner scnr = new Scanner(System.in);

    public UserControllerforUser() {}

    //로그인 후 id저장을 위해서 필요
    private String userId;
    public String getUserId() { return userId; }

    //회원가입
    public void signUp()..

    //사용자 로그인
    public void login(){    ..

    //사용자 로그아웃(프로그램 종료)
    public void logout() {..
}
```

```java
public class AdminController {
    Admin admin = new Admin();
    Scanner scnr = new Scanner(System.in);

    private boolean isAdmin = false;

    public boolean getIsAdmin() {..

    //관리자 로그인
    public void login() {..

    //관리자 로그아웃(프로그램 종료)
    public void logout() {..
}
```

- **View**

: 리스트를 받아서 출력하는 함수와 페이지(로그인 및 회원가입 화면과, 관리자와 사용자의 기능 선택지를 보여주는 화면)

```
public class View {
    //검색된 책 목록이나 전체 책 목록 출력
    public void printBookList(List<Book> bookList, String type){

    //사용자가 등록한 책 목록 출력 (검색된 책 목록과 전체 책 목록 출력이랑 다르게 한 이유는 사용자가 등록한 책 목록을 출력 시에는 사용자 id가 없어도 됨)
    public void printUsrBookList(List<Book> usrBookList) {

    //비활성화 상태의 사용자 목록이나 전체 사용자 목록 출력
    public void printUserList(List<User> UserList, String type)

    //market에서 필요한 페이지 출력
    public void printFirstPage() {

    public void printSecondPageforAdmin() {

    public void printSecondPageforUser() {

}
```

- **Main 과 전체적인 시스템의 흐름을 관리하는 Market 클래스** (market class도 controller 부분에 속한다.)

```
public class Main {
    public static void main(String[] args) {
        //드라이버로딩
        try {
            Class.forName("com.mysql.jdbc.Driver");
        }
        catch(Exception e) {
            e.printStackTrace();
        }

        Market market = new Market();
        market.marketOpen();
    }
}
```

```
public class Market {
    Scanner scnr = new Scanner(System.in);

    UserControllerforUser userControllerforUser = new UserControllerforUser();
    UserControllerforAdmin userControllerforAdmin = new UserControllerforAdmin();
    BookControllerforUser bookControllerforUser = new BookControllerforUser();
    BookControllerforAdmin bookControllerforAdmin = new BookControllerforAdmin();
    AdminController adminController = new AdminController();
    View view = new View();

    private String userId;
    private boolean isAdmin = false;

    protected void marketOpen() {

    private void secondPage() {


}
```

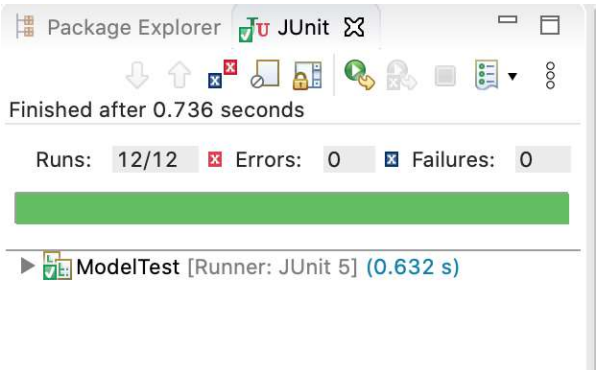
3. 테스트 케이스 및 테스트 결과

View에서는 출력을 하고, controller에서는 사용자의 입력을 받아 data의 직접적인 부분은 model쪽에서 처리하도록 넘겨주기 때문에 실제로 데이터를 다루는 핵심기능은 model에서 이루어진다고 볼 수 있다. 따라서 model의 기능을 하는 클래스들을 테스트하였다. 그 중 관련이 있는 함수들은 같이 묶어서 테스트하였다.

테스트를 시작하기에 앞서, 정확하게 테스트를 하기 위해서 테스트는 데이터베이스가 비어있는 상태에서 해야 한다. 또한 각각의 테스트 함수를 개별적으로 처리하기 위해서는 함수가 끝날 때에는 데이터베이스에 데이터가 남아있지 않도록 해야 한다. 따라서 user와 관련된 데이터를 처리하는 함수들은 그 함수에서 데이터베이스에 추가하고 처리했던 데이터들을 함수가 끝날 때 delete하는 함수를 이용하여 데이터베이스를 비워주었다. user와 관련된 데이터를 처리하는 함수는 이러한 방법으로 가능했지만, book과 관련된 데이터를 처리하는 일부 함수들은 함수의 parameter로 원래 사용자가 입력하던 책의 unique한 key값을 받는데, 위에서 언급했듯이 key가 데이터베이스에서 auto increment처리되어있으므로 key값을 함수에 직접 넣어 테스트를 하기 위해서는 매 함수마다 book table을 truncate해주어야 한다. (truncate을 해주지 않으면, 데이터베이스에서의 key값은 계속해서 증가하므로, 함수의 parameter로 어떤 key값을 넣어주어야 하는지 예측할 수 없다.)

```
class ModelTest {

    BookModel bookModel = new BookModel();
    UserModel userModel = new UserModel();
    Book book1 = new Book("sametitle", "000000", "author1", "publisher1", "2020", "user1", "10000", "good");
    Book book2 = new Book("sametitle", "000001", "author2", "publisher2", "2020", "user2", "20000", "fair");
    Book book3 = new Book("only title", null, null, null, null, null, null, null);
    User user1 = new User("name1", "user1", "pw1", "01000000000", "@google.com");
    User user2 = new User("name2", "user2", "pw2", "01011111111", "@naver.com");
    List<Book> bookList = new ArrayList<Book>();
    List<User> userList = new ArrayList<User>();

    //user
    void testAddandDeleteUser() {

    void testIsThereSearchedUser() {

    void testIsThereUserWthIdPw() {
    //convertUserState, setDeactivatedUserList, isUserDeactivated, isThereDeactivatedUser
    void testDeactivatedUser() {

    void testSetTotalUserList() {

    void testDeleteBookListWthUsrId() {

    //book
    void testAddandDeleteBook() {

    void testSetTotalBookList() {

    //setSearchedBookList, isThereSearchedBook
    void testSearchedBook() {
    //isThereUserOwnBook, getUserOwnBookList
    void testIsThereUserOwnBook() {

    void testModifyBookInfo() {

    //checkBuyBookIntention, getBookSellerID, getBookSellerEmail, getBuyerEmail
    void testBuyBook() {

}
```

```
}
//convertUserState, setDeactivatedUserList, isUserDeactivated, isThereDeactivatedUser
@Test
void testDeactivatedUser() {
    userModel.addUser(user1);
    assertFalse(userModel.isThereDeactivatedUser()); //사용자 초기상태는 activated
    assertFalse(userModel.isUserDeactivated(user1.getId()));
    userModel.convertUserState(user1.getId());
    assertTrue(userModel.isThereDeactivatedUser());
    assertTrue(userModel.isUserDeactivated(user1.getId()));
    userModel.setDeactivatedUserList();
    assertEquals(userModel.deactivatedUserList.get(0).getId(), user1.getId());
    userModel.deleteUser(user1.getId());
}

//checkBuyBookIntention, getBookSellerEmail, getBookSellerID, getBuyerEmail
@Test
void testBuyBook() {
    bookModel.truncateBookTable();
    userModel.addUser(user1);
    userModel.addUser(user2);
    bookModel.addBook(book1);//user1이 책을 등록
    bookModel.setTotalBookList();
    assertEquals("N", bookModel.totalBookList.get(0).getPurchaseIntention());
    bookModel.checkBuyBookIntention(1); //key=1
    bookModel.setTotalBookList();
    assertEquals(user1.getId(), bookModel.getBookSellerID(1)); //key=1
    assertEquals(user1.getEmail(), bookModel.getBookSellerEmail(1)); //key=1
    assertEquals(user2.getEmail(), bookModel.getBuyerEmail(user2.getId()));//user2가 책을 산다고 가정
    assertEquals("Y", bookModel.totalBookList.get(0).getPurchaseIntention());
    userModel.deleteUser(user1.getId());
    userModel.deleteUser(user2.getId());
}
```



▶12개의 테스트함수가 에러없이 실행되었다.