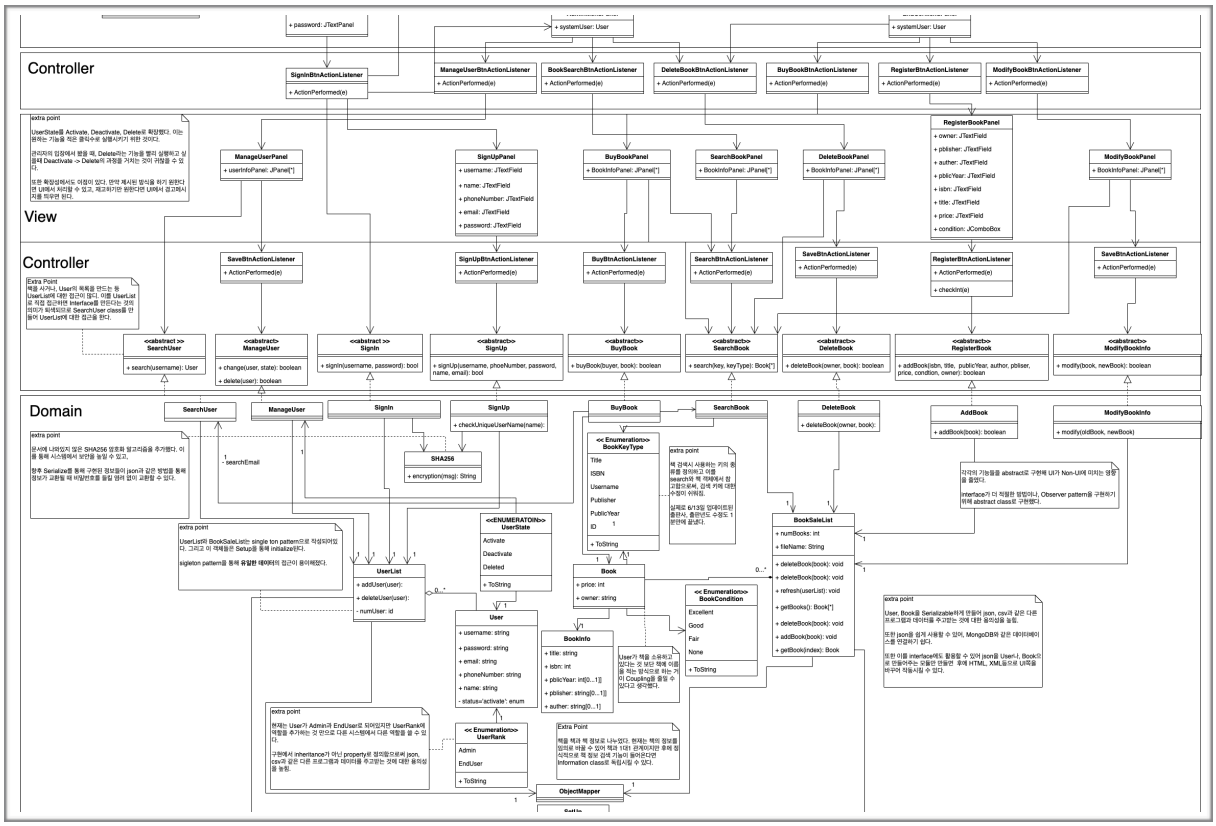
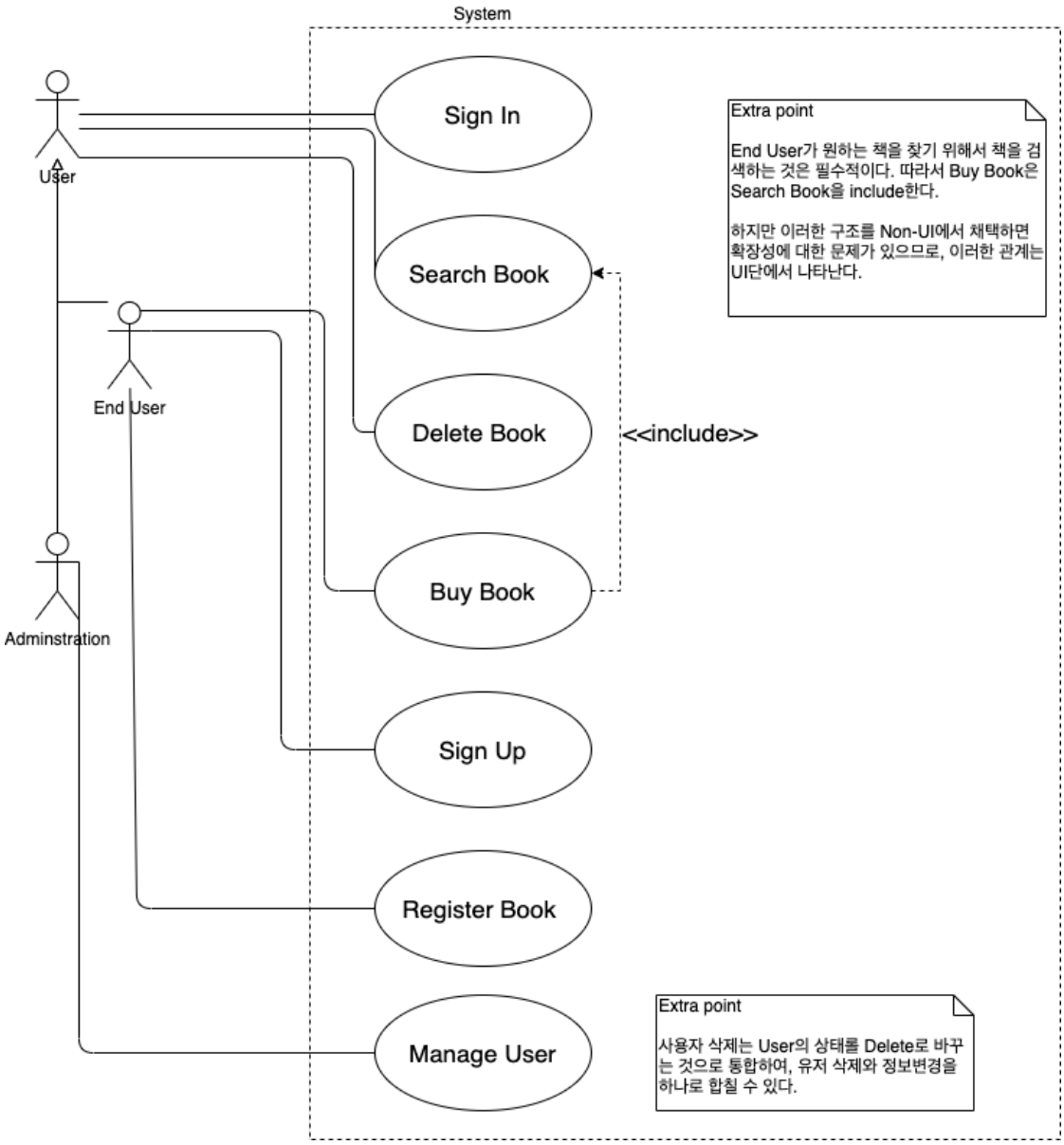
소프트웨어공학 과제

헌책 장터 시스템



20182592 장원범

Use-case Diagram



End-user Use-case 명세서

End-user Use case UC1: Buy Book

Main Success Scenario

1. 사용자가 책 검색화면으로 이동한다.
2. 사용자가 제목, ISBN 번호, 저자, 판매자 id 중 하나를 입력한다.
3. 검색에 해당하는 책 의 제목, 출판사, 저자 정보, 출판년도, 가격과 상태가 출력된다.
4. 구매하고자 하는 책을 선택한다.
5. 구매하고자 하는 책의 주인에게 메일을 보낸다.

사용자는 구매하고 싶은 책을 모두 구매할때까지 2~4를 반복한다.

6. 사용자가 구매를 끝낸다.

Extensions

*a. 시스템이 실패할 경우
    1. 시스템을 재시작 한다.

2a. 키워드를 입력 안한 채 검색을 하는 경우
    1. 키워드를 입력하지 않았다는 메시지를 출력한다.
    2. 책 검색화면으로 넘어간다.

3a. 검색한 키워드에 해당하는 책이 없는 경우
    1. 사용자에게 책이 없다는 메시지를 출력한다.
    2. 책을 검색하는 화면으로 돌아간다.

4a. 구매하고자 하는 책의 주인의 상태가 deactivate일 경우
    1. 해당 사용자는 비활성화중이라는 메시지를 출력한다.
    2. 해당 사용자를 제외한 책 목록을 다시 출력한다.

End-user Use case UC2: Register Book

Main Success Scenario

1. 사용자가 책 등록 화면으로 이동한다.
2. 책의 제목, 출판사, 저자 정보, 출판년도, 가격 및 상태를 입력한다.
3. 책이 시스템에 등록된다.

Extensions

*a. 시스템이 실패할 경우
    1. 시스템을 재시작 한다.

2. 책의 제목이 입력이 안되어있을 경우
    1. 책 제목이 입력이 안되어있다는 메시지를 출력한다.

2. 책 등록 화면으로 돌아간다.

Administration Use-case 명세서

End-user Use case UC1:Delete User

Main Success Scenario

1. 사용자 관리 화면으로 이동한다.
2. 모든 사용자의 목록이 출력된다.
3. 사용자를 삭제한다.
4. 변경된 사용자 목록이 출력된다.

원하는 모든 사용자가 삭제될때 까지 2~4번을 반복한다.

5. 사용자 삭제를 끝낸다.

Extensions

*a. 시스템이 실패할 경우
    1. 시스템을 재시작 한다.

3a. 사용자의 상태가 activate인 경우
    1. 사용자의 상태가 activate라는 메시지를 출력한다.
    2. 사용자 2로 돌아간다.

End-user Use case UC2: Delete Book

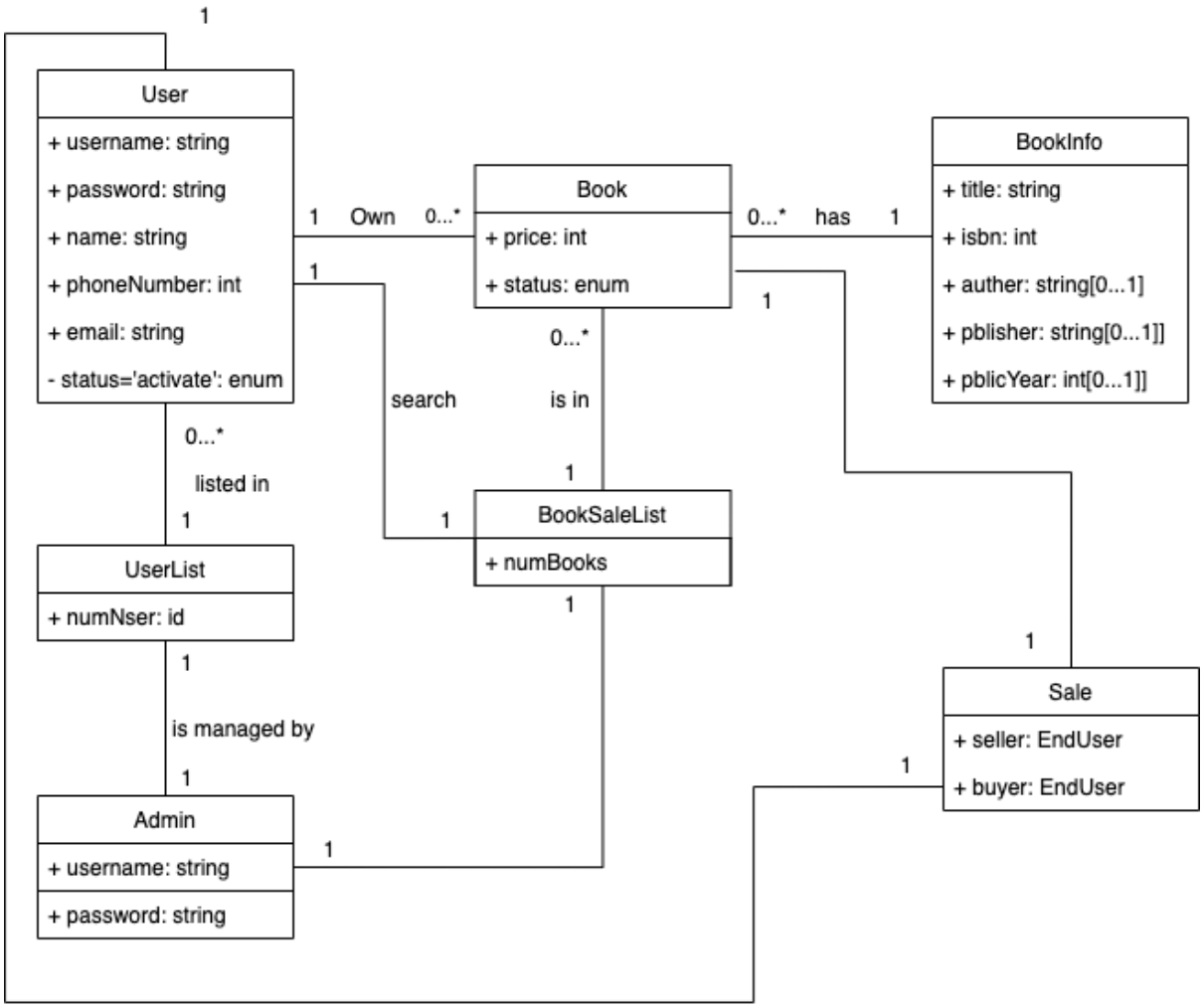
Main Success Scenario

1. 책 관리 화면으로 이동한다.
2. 접근할 수 있는 모든 책의 목록을 출력한다.
3. 책을 체크한 후 삭제 버튼을 누른다
4. 책 삭제를 종료한다.

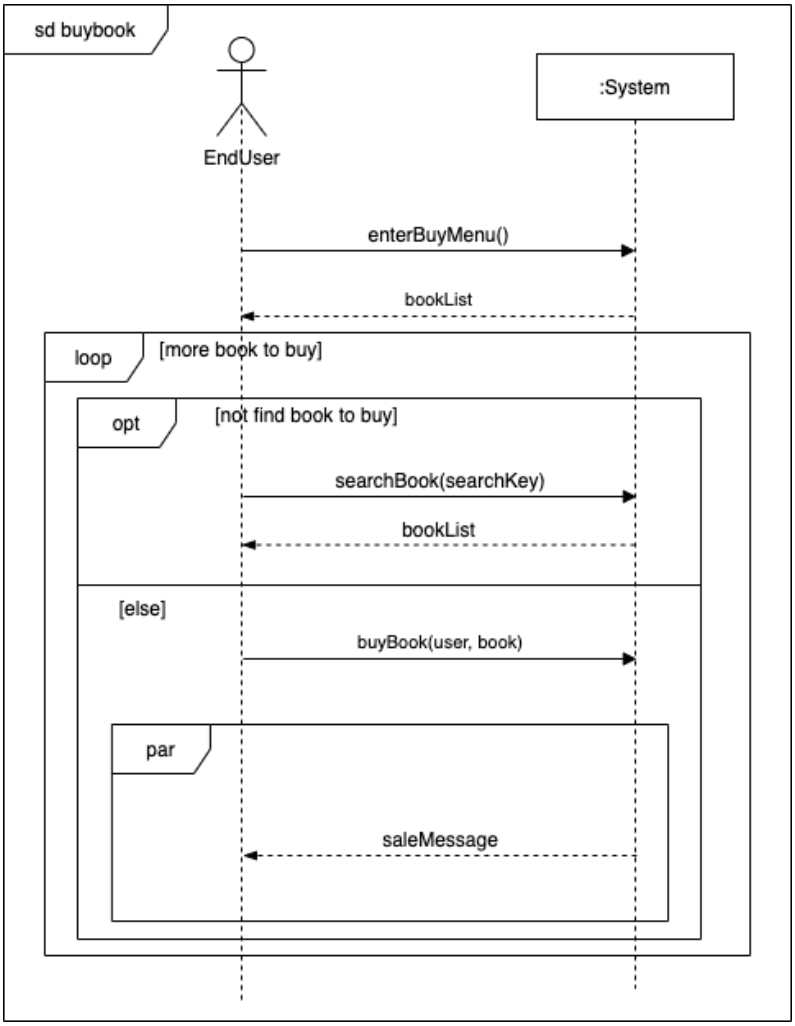
Extensions

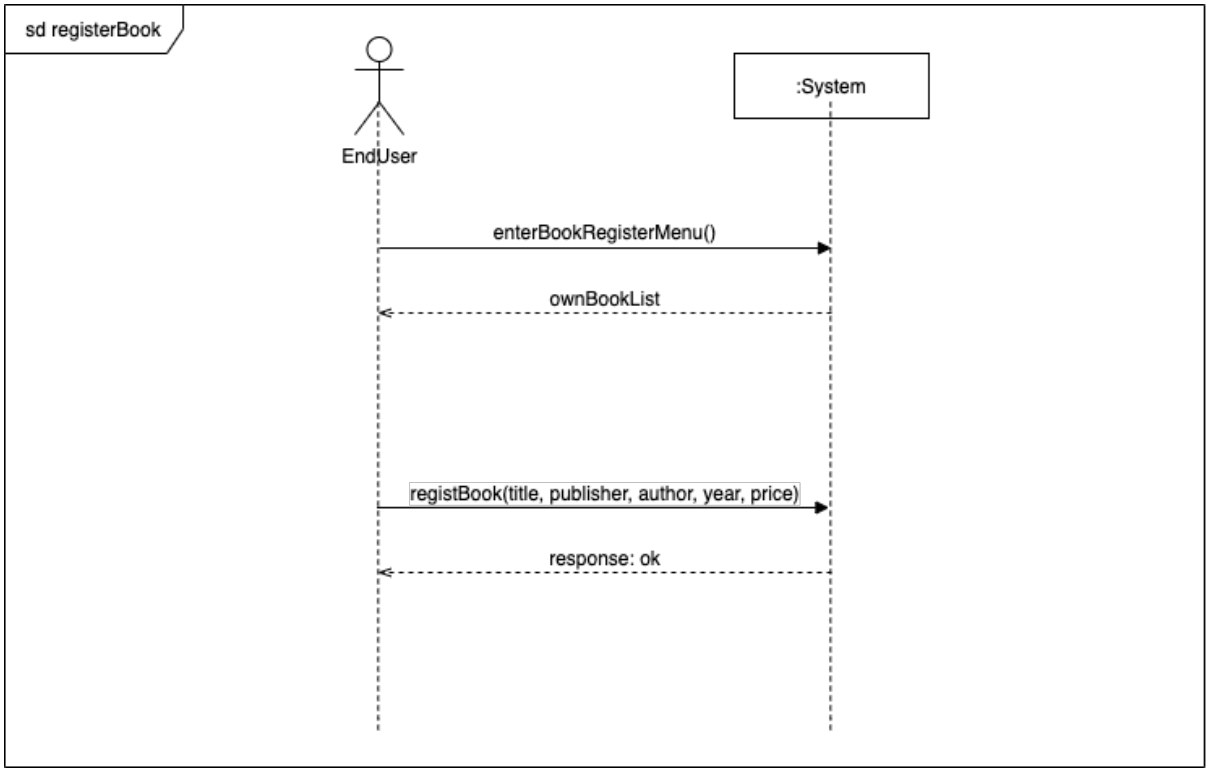
*a. 시스템이 실패할 경우
    1. 시스템을 재시작 한다

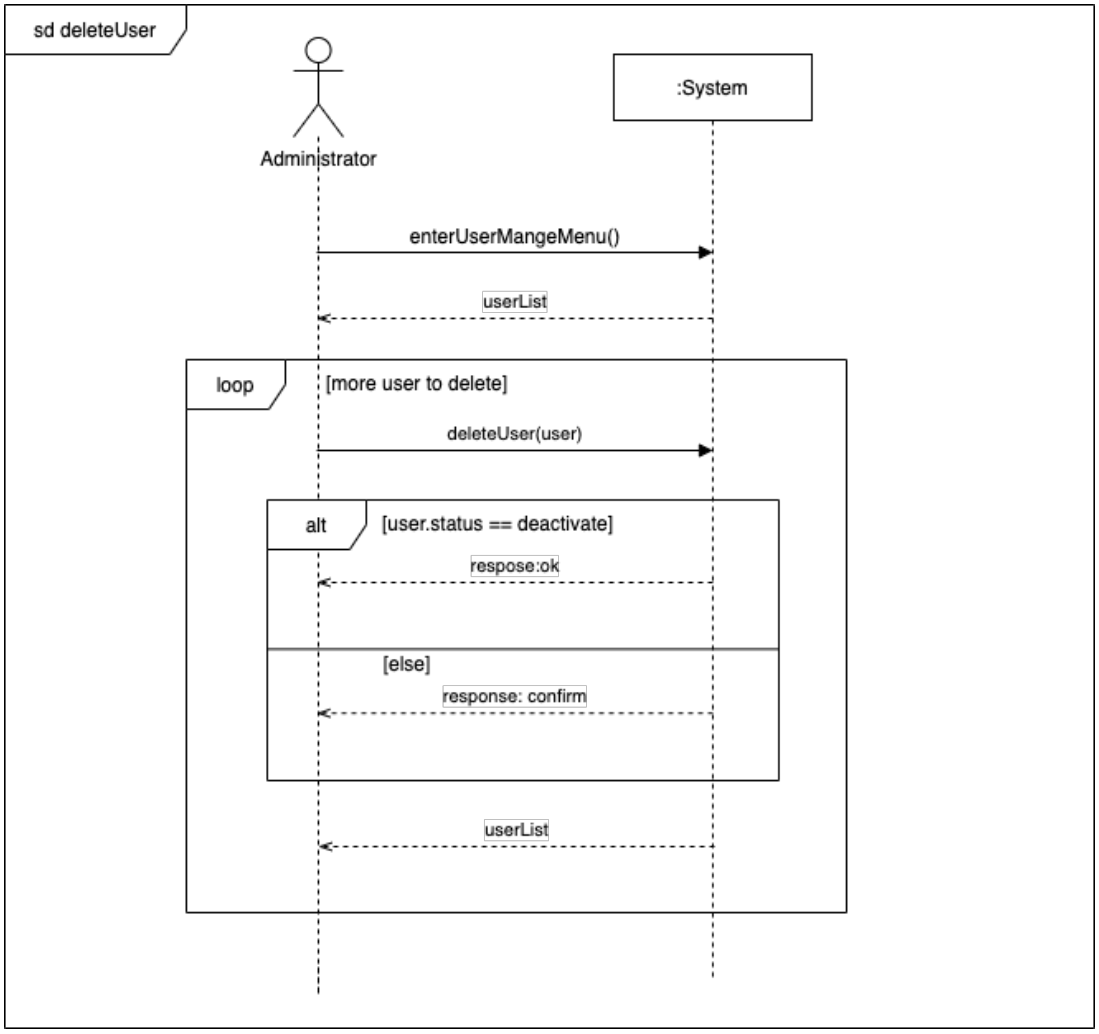
Domain Model

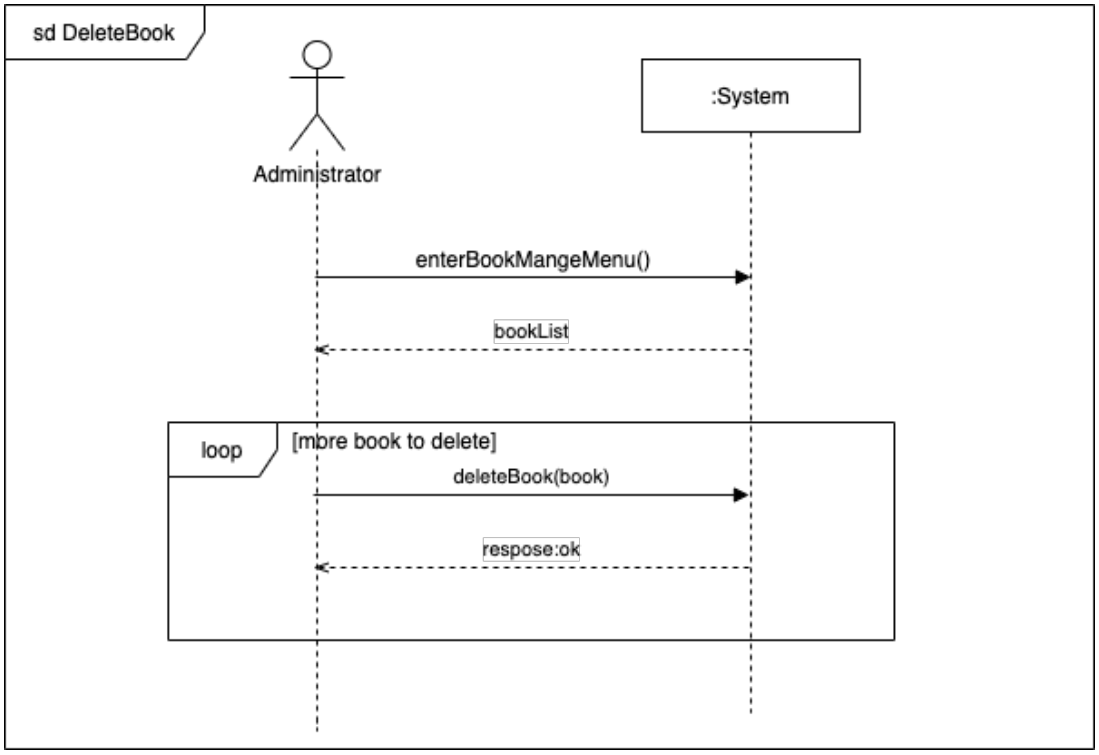


System Sequence Diagram






sd deleteUser
Administrator
:System
enterUserMangeMenu()
userList
loop
[more user to delete]
deleteUser(user)
alt
[user.status == deactivate]
respose:ok
[else]
response: confirm
userList

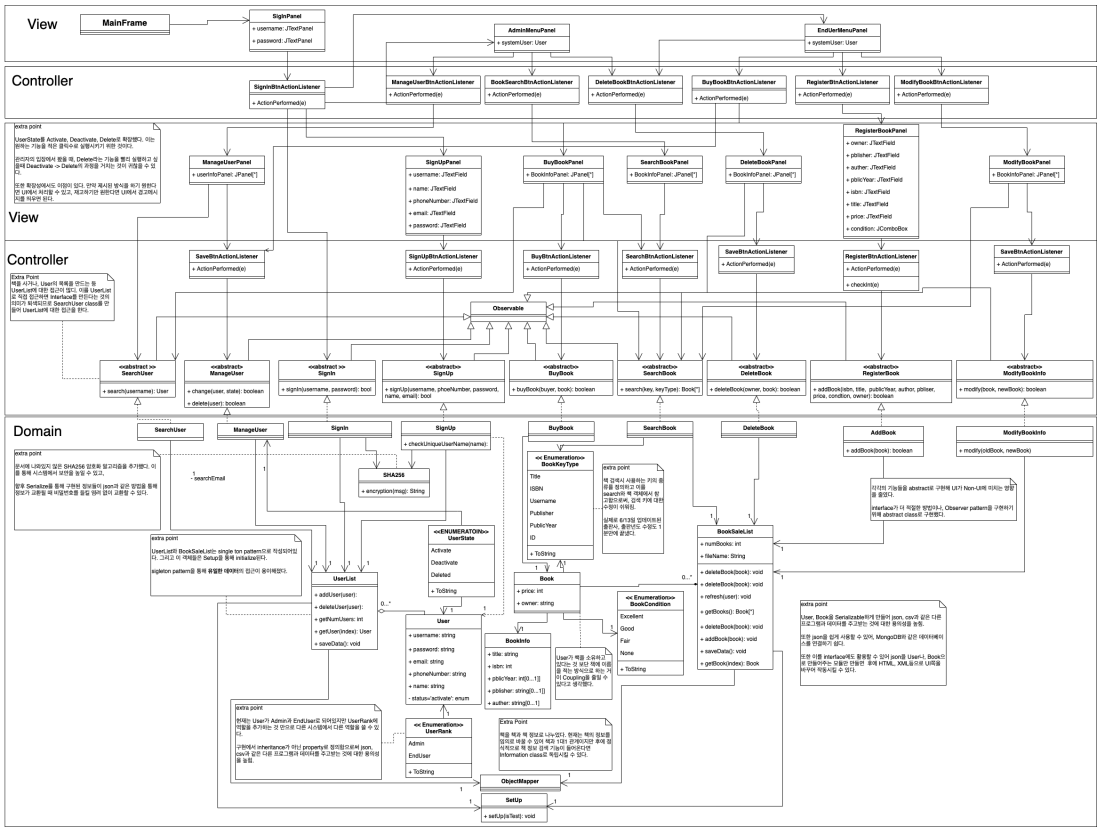

sd DeleteBook
Administrator
:System
enterBookMangeMenu()
bookList
loop
[more book to delete]
deleteBook(book)
respose:ok

Operation Contract

Operation	registerBook(title, publisher, author, year, price)
Cross References	End-user Use case UC2: Register Book
Preconditions	BookRegisterMenu에 있어야함.
Postconditions	- Book객체가 생성되고 받은 parameter로 초기화가 되었다. - Book객체가 BookSaleList에 연결 되었다. - BookInfo객체가 생성되고 주어진 parameter로 초기화 되며, Book에 연결 되었다.

Operation	buyBook()
Cross References	End-user Use case UC2: Register Book
Preconditions	Book과, 사는 사람의 객체가 생성되어있다..
Postconditions	- Book의 owner가 사는 사람의 username으로 변경되었다. - BookSaleList가 업데이트 되었고, 그 정보가 디스크에 저장되었다.

Class Diagram



전체적인 구조

UI와 Non-UI의 의존도를 낮추기 위해 MVC pattern을 사용했다. UI와 Non-UI는 abstract class를 통해 message를 주고 받고, 각각의 abstract class는 observable을 상속받는다.

UI쪽 Class가 많아 UI쪽은 간단하게 나타냈다.

GRAPS Pattern

1. Use-case Controller pattern
   1.1.Use-case에 사용되는 controller가 7개이다. 이를 1개나 2개의 Object에 responsibility 부여할 경우 한 class의 크기가 커질 것이라고 생각했다. 따라서 각각의 Use-case마다 abstract class로 controller를 만들었다.

2. Low Coupling pattern
   2.1. RegisterBook이 Book을 생성하는 Creator가 된다면 RegisterBook이 Book을 Create하고 생성된 Book을 BookList에 넣어야한다. 이때 RegisterBook, BookList, Book사이의 원형 dependency가 생긴다. RegisterBook이 BookList에게 Book생성의 책임을 전가함으로써 dependency를 줄였다.
   2.2. SignUp도 위와 같은 이유로 UserList에 책임을 전가했다.

3. Pure Fabric Pattern
   3.1.UI에서나 책 거래에서나 UserList에서 User를 찾는 과정이 빈번히 일어날 것으로 예상된다. 현재는 username으로만 User를 검색하게 되지만 후에는 UserName뿐만 아니라 email이나 전화번호로 User를 찾을 경우도 생길 것이다. 따라서 Domain Model이나 Use-Case diagram에서는 안나왔지만

4. Indirection Pattern
   4.1. SignUp이 UserList에게 User를 create하고 link를 만들기 때문에 SingUp이 필요 없을 수 있다. 하지만 비밀번호 암호화를 하기 위해 SHA256을 사용했고, UserList에서는 많은 Coupling이 존재한다. 따라서 SHA256과 Coupling을 만들고 싶지 않아 SignUp을 추가했다.
5. Protection Variable Pattern
   5.1. UI쪽 Class는 많이 변경된다. 이의 영향을 줄이기 위해 View + Controller는 abstract class를 통해 Domain Layer와 message를 주고 받는다. 이러한 패턴에서는 interface가 더 적합하나 Observer Pattern을 쓰기 때문에 abstract class로 설계할수 밖에 없었다. 이를 극복하기 위해 각각의 abstract class의 함수는 observable에서 구현된 함수를 제외하고 모두 abstract function이다.

Solid Pattern

1. Dependency Inversion Principle
   1.1. 본 시스템은 크게 사용자 관리, 책 관리로 나눠져있다. 즉 각각을 모듈로 볼 수 있다. 따라서 전체 프로그램은 이 모듈들의 변화에 민감해서는 안된다. 따라서 각각의 모듈을 abstract class로 감싸 프로그램을 보호한다.

2. Interface Segregation Principle
   2.1. Admin과 EndUser는 서로 다른 기능을 사용한다. 각각이 사용하는 abstract class는 불필요한 Admin과 EndUser의 Coupling을 만들지 말아야 한다. 이를 위해 본 설계에서는 각각의 기능들 별로 Abstract를 만들었고 이들의 Composition으로 사용자에게 기능을 제공할 수 있다.
   2.2. Inheritance보다 Composition을 사용하기 위해 Admin Interface와 EndUser Interface를 따로 만들지 않았다.

Extra Point

1. UserRank
   1.1. Admin과 EndUser는 모두 User라는 관점에서 하나의 Abstract class로 일반화 될 수 있다. Admin과 EndUser의 역할은 공통적인 기능이 없고, 접근수준을 제한한 것이라 상속으로 문제를 해결한 다면 typecastring을 해야하는 등 여러가지 문제점이 발생한다. 또한 Json으로 통신할 때, 이 객체가 Admin인지, EndUser인지 알기가 어렵다. 따라서 UserRank로 접근수준을 제한하는 방법으로 Admin과 EndUser를 구분했다.

2. BookKeyType

2.1. 책을 검색할 때 무엇을 기준으로 검색할 것인지를 정하는 역할로 BookKeyType을 정의하고 SearchBook에서 BookKeyType을 참조하여 검색하게 했다. 이로인해 검색하는 객체의 코드 수정이 쉬워졌다. 실제로 6/13일에 업데이트 된 출산사, 출판년도로 검색하는 부분도 1분만에 코드수정을 완료했다.

3. BookID
   3.1. 사용자들은 같은 정보를 같이 책을 여러개를 소유하고 있다. 그렇기 때문에 UI Layer를 Java가 아닌 HTML, React등을 쓸 경우 그 객체의 참조값을 알기 힘들어, 어떠한 책이 수정되었는지 알기 힘들다. 이러한 시스템의 확장성을 위해 BookID로 Book의 Primary key를 만들었다.

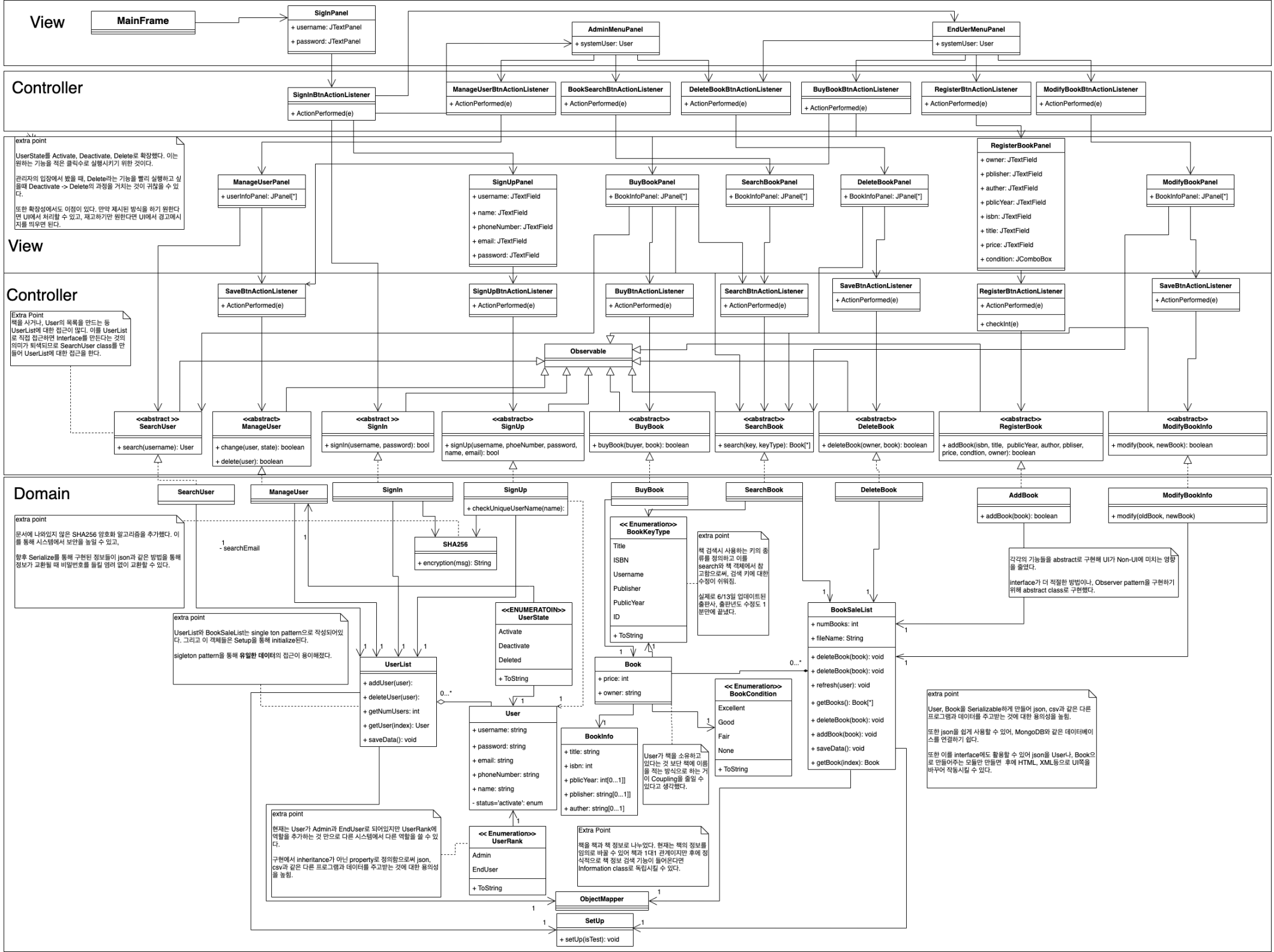
4. Sigleton Pattern
   4.1. UserList와 BookList를 전체 시스템에서 유일하게 존재해야한다. 만약 UserList와 BookList의 여러개의 객체를 생성할 경우 각각의 데이터를 동기화 하는 문제가 있고, 개발자의 실수가 증가한다. 따라서 UserList와 BookList에 singleton pattern을 적용해 한 시스템 내에서 하나의 객체만을 생성할 수 있게 되었고, 안정적으로 데이터를 다룰 수 있게 되었다. 만약 후에 여러 유저가 동시에 이 시스템을 사용하게 된다면, lock을 걸어 상호배제적으로 정보를 수정하게 할 수 있다.

5. SHA256 암호화 알고리즘
   5.1. 본 설계의 목적은 UI와 Non-UI의 분리에 있다. 따라서 통신을 통해서 정보를 주고 받을 수 있다고 생각해, 민감한 정보인 비밀번호를 SHA256을 통해 암호화 했다.

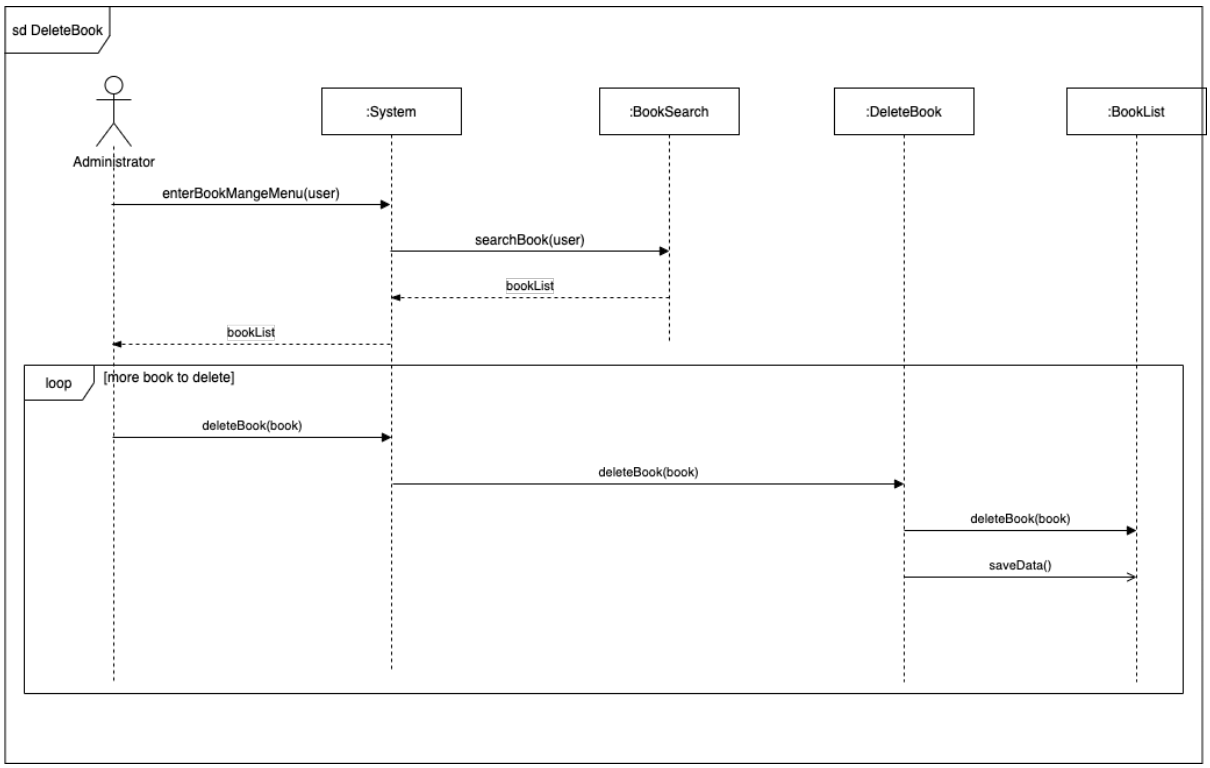
6. SearchUser
   6.1. ManageUser의 UI를 구성할 때 사용자의 정보들이 필요하다. 만약 UI쪽에서 abstract class가 아닌 UserList을 통해 정보를 직접 접근하게 된다면 UI와 Non-UI의 의존도가 높아져 유지보수하기 힘들어진다. 따라서 SearchUser를 통해 User에 대한 정보를 접근할 수 있다.

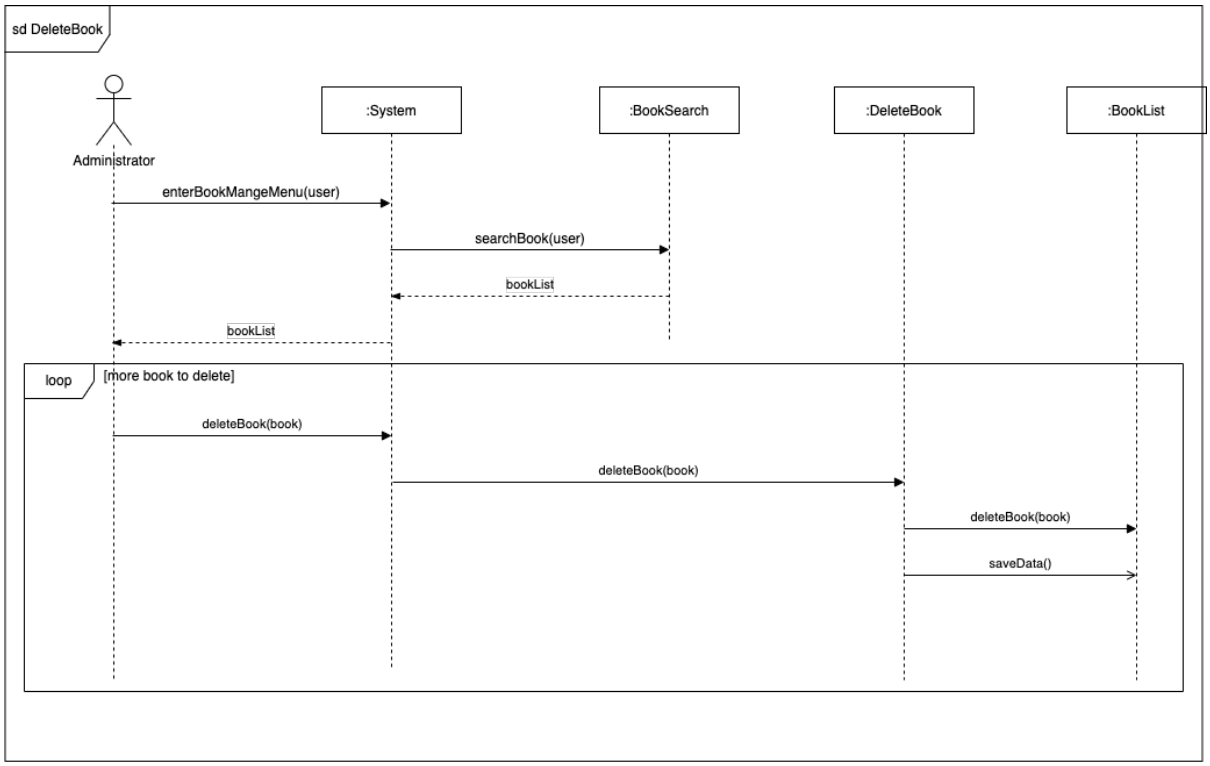
7. UserState
   7.1. User의 정보를 수정하거나 User를 삭제하는 기능을 분리시키지 않고 UserState를 Activate, Deactivate, Delete 세가지 상태로 만들어 통합시켰다. 이를 통해 관리자가 사용자를 잘못 삭제하는 것을 방지하기 위해 Delete하고 1시간 후에 내용을 적용하는 등 다양한 확장이 가능해졌다.
   7.2. 문서에서 Activate된 사용자는 삭제하지 말라는 명시가 없어 관리자에게 확인하는 메시지를 띄우는 식으로 프로그램을 구성했으나, 삭제를 하지 못하게 만드는 등의 수정을 Domain의 코드 수정 없이 할 수 있다.

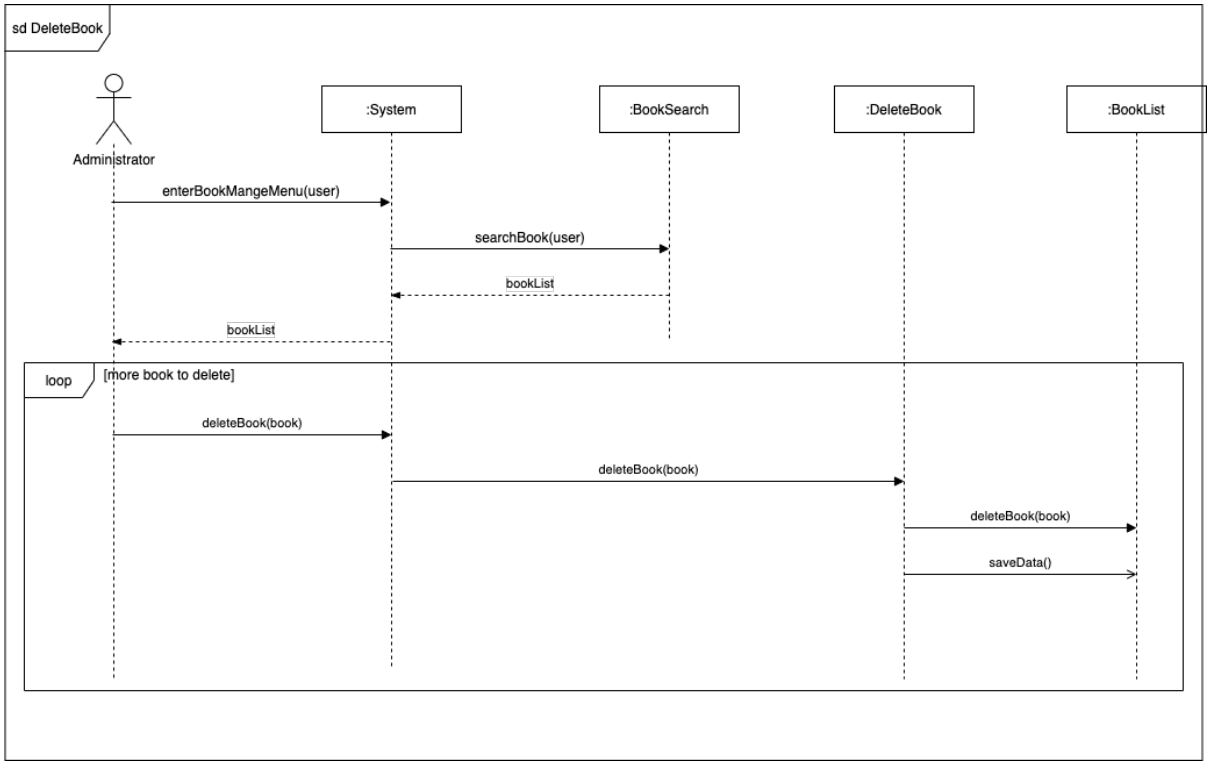
8. User, Book의 Serializable
   8.1. User와 Book을 serializable하게 만들어줌으로써, Jackson을 통해 json객체으로 변환 할 수 있게 했다. 이를 통해 HTML, XML뿐만 아니라 MongoDB와 같은 시스템에서도 이 class들을 처리할 수 있게되었고, Domain의 interface에서 간단한 처리만 해준다면 서버에 올려서 원격으로 데이터를 처리하게 할 수 있는 등의 확장을 할 수 있다.


View
MainFrame
SigInPanel
+ username: JTextPanel
+ password: JTextPanel
AdminMenuPanel
+ systemUser: User
EndUerMenuPanel
+ systemUser: User
Controller
SignInBtnActionListener
+ ActionPerformed(e)
ManageUserBtnActionListener
+ ActionPerformed(e)
BookSearchBtnActionListener
+ ActionPerformed(e)
DeleteBookBtnActionListener
+ ActionPerformed(e)
BuyBookBtnActionListener
+ ActionPerformed(e)
RegisterBtnActionListener
+ ActionPerformed(e)
ModifyBookBtnActionListener
+ ActionPerformed(e)
extra point
UserState를 Activate, Deactivate, Delete로 확장했다. 이는 원하는 기능을 적은 클릭수로 실행시키기 위한 것이다.
관리자의 입장에서 봤을 때, Delete라는 기능을 빨리 실행하고 싶을때 Deactivate -> Delete의 과정을 거치는 것이 귀찮을 수 있다.
또한 확장성에서도 이점이 있다. 만약 제시된 방식을 하기 원한다면 UI에서 처리할 수 있고, 재고하기만 원한다면 UI에서 경고메시지를 띄우면 된다.
View
ManageUserPanel
+ userInfoPanel: JPanel[*]
SignUpPanel
+ username: JTextField
+ name: JTextField
+ phoneNumber: JTextField
+ email: JTextField
+ password: JTextField
BuyBookPanel
+ BookInfoPanel: JPanel[*]
SearchBookPanel
+ BookInfoPanel: JPanel[*]
DeleteBookPanel
+ BookInfoPanel: JPanel[*]
RegisterBookPanel
+ owner: JTextField
+ pblisher: JTextField
+ auther: JTextField
+ pblicYear: JTextField
+ isbn: JTextField
+ title: JTextField
+ price: JTextField
+ condition: JComboBox
ModifyBookPanel
+ BookInfoPanel: JPanel[*]
Controller
SaveBtnActionListener
+ ActionPerformed(e)
SignUpBtnActionListener
+ ActionPerformed(e)
BuyBtnActionListener
+ ActionPerformed(e)
SearchBtnActionListener
+ ActionPerformed(e)
SaveBtnActionListener
+ ActionPerformed(e)
RegisterBtnActionListener
+ ActionPerformed(e)
+ checkInt(e)
SaveBtnActionListener
+ ActionPerformed(e)
Extra Point
책을 사거나, User의 목록을 만드는 등 UserList에 대한 접근이 많다. 이를 UserList로 직접 접근하면 Interface를 만든다는 것의 의미가 퇴색되므로 SearchUser class를 만들어 UserList에 대한 접근을 한다.
Observable
<<abstract >> SearchUser
+ search(username): User
<<abstract>> ManageUser
+ change(user, state): boolean
+ delete(user): boolean
<<abstract >> SignIn
+ signIn(username, password): bool
<<abstract>> SignUp
+ signUp(username, phoeNumber, password, name, email): bool
<<abstract>> BuyBook
+ buyBook(buyer, book): boolean
<<abstract>> SearchBook
+ search(key, keyType): Book[*]
<<abstract>> DeleteBook
+ deleteBook(owner, book): boolean
<<abstract>> RegisterBook
+ addBook(isbn, title, publicYear, author, pbliser, price, condtion, owner): boolean
<<abstract>> ModifyBookInfo
+ modify(book, newBook): boolean
Domain
SearchUser
ManageUser
SignIn
SignUp
+ checkUniqueUserName(name):
BuyBook
SearchBook
DeleteBook
AddBook
+ addBook(book): boolean
ModifyBookInfo
+ modify(oldBook, newBook)
extra point
문서에 나와있지 않은 SHA256 암호화 알고리즘을 추가했다. 이를 통해 시스템에서 보안을 높일 수 있고,
향후 Serialize를 통해 구현된 정보들이 json과 같은 방법을 통해 정보가 교환될 때 비밀번호를 들킬 염려 없이 교환할 수 있다.
1
- searchEmail
SHA256
+ encryption(msg): String
각각의 기능들을 abstract로 구현해 UI가 Non-UI에 미치는 영향을 줄였다.
interface가 더 적절한 방법이나, Observer pattern을 구현하기 위해 abstract class로 구현했다.
<< Enumeration>> BookKeyType
Title
ISBN
Username
Publisher
PublicYear
ID
+ ToString
extra point
책 검색시 사용하는 키의 종류를 정의하고 이를 search와 책 객체에서 참고함으로써, 검색 키에 대한 수정이 쉬워짐.
실제로 6/13일 업데이트된 출판사, 출판년도 수정도 1분만에 끝났다.
extra point
UserList와 BookSaleList는 single ton pattern으로 작성되어있다. 그리고 이 객체들은 Setup을 통해 initialize된다.
sigleton pattern을 통해 유일한 데이터의 접근이 용이해졌다.
<<ENUMERATOIN>> UserState
Activate
Deactivate
Deleted
+ ToString
BookSaleList
+ numBooks: int
+ fileName: String
+ deleteBook(book): void
+ deleteBook(book): void
+ refresh(user): void
+ getBooks(): Book[*]
+ deleteBook(book): void
+ addBook(book): void
+ saveData(): void
+ getBook(index): Book
UserList
+ addUser(user):
+ deleteUser(user):
+ getNumUsers: int
+ getUser(index): User
+ saveData(): void
Book
+ price: int
+ owner: string
<< Enumeration>> BookCondition
Excellent
Good
Fair
None
+ ToString
User
+ username: string
+ password: string
+ email: string
+ phoneNumber: string
+ name: string
- status='activate': enum
BookInfo
+ title: string
+ isbn: int
+ pblicYear: int[0...1]]
+ pblisher: string[0...1]]
+ auther: string[0...1]
User가 책을 소유하고 있다는 것 보단 책에 이름을 적는 방식으로 하는 거이 Coupling을 줄일 수 있다고 생각했다.
extra point
User, Book을 Serializable하게 만들어 json, csv과 같은 다른 프로그램과 데이터를 주고받는 것에 대한 용의성을 높힘.
또한 json을 쉽게 사용할 수 있어, MongoDB와 같은 데이터베이스를 연결하기 쉽다.
또한 이를 interface에도 활용할 수 있어 json을 User나, Book으로 만들어주는 모듈만 만들면 후에 HTML, XML등으로 UI쪽을 바꾸어 작동시킬 수 있다.
extra point
현재는 User가 Admin과 EndUser로 되어있지만 UserRank에 역할을 추가하는 것 만으로 다른 시스템에서 다른 역할을 쓸 수 있다.
구현에서 inheritance가 아닌 property로 정의함으로써 json, csv과 같은 다른 프로그램과 데이터를 주고받는 것에 대한 용의성을 높힘.
<< Enumeration>> UserRank
Admin
EndUser
+ ToString
Extra Point
책을 책과 책 정보로 나누었다. 현재는 책의 정보를 임의로 바꿀 수 있어 책과 1대1 관계이지만 후에 정식적으로 책 정보 검색 기능이 들어온다면 Information class로 독립시킬 수 있다.
ObjectMapper
SetUp
+ setUp(isTest): void

Sequence Diagram






sd DeleteBook
Administrator
:System
:BookSearch
:DeleteBook
:BookList
enterBookMangeMenu(user)
searchBook(user)
bookList
bookList
loop
[more book to delete]
deleteBook(book)
deleteBook(book)
deleteBook(book)
saveData()