CSCI-5448 OOAD

BOOKWISE - Project Part 4

Team: Aayush Grover, Alimulla Shaik, Shuzhe Zhang, Zhihao Wu

1. What features were implemented and a class diagram showing the final set of classes and relationships of the system. (This may have changed from what you originally anticipated from earlier submissions). Discuss what changed in your class diagram and why it changed, or how it helped doing the diagrams first before coding if you did not need to change much.

Features implemented:

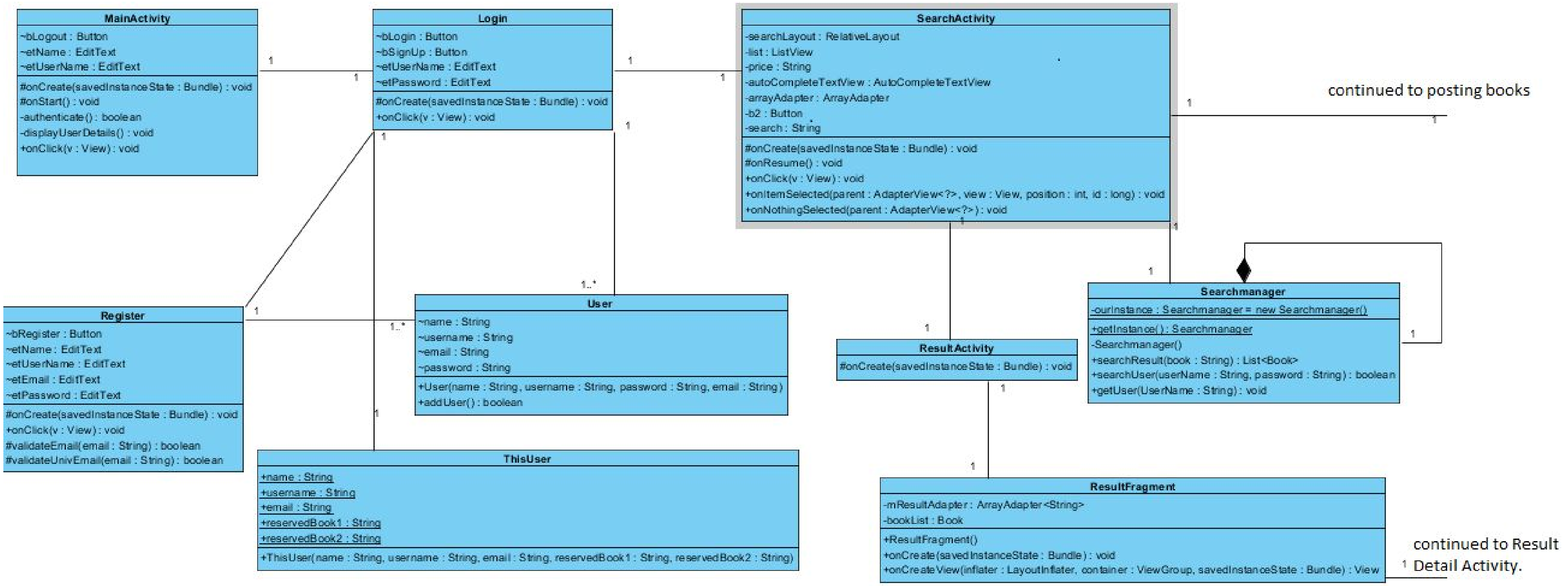
- User Login which allows user to login into his/her account
- Register Page to allow users to create an account
- Seller can add/post book details and provides email id to connect with interested buyers
- Buyer can able to search a particular book he is interested in and able to check current status of the book availability.
- If the book is available then buyer can reserve the book and all reserved books are listed in search activity.

Benefits of designing before coding:

- It is always best practice to design before coding. As design will help to understand high level view of the system, it will be easier to code or implement later.
- We can review design at any time for errors before starting implementation and can update it in case of any changes.
- This will improve quality of the product and makes developers’ job a lot easier during implementation phase.
- Design document will provide details of the system so anyone can learn from it without prior knowledge of the topic.
- If there are any changes in design in future then it can be incorporated easily and not difficult to implement these changes.

Final Class diagram is shown below and please maximize it for better view as it wasn’t fitting in page.

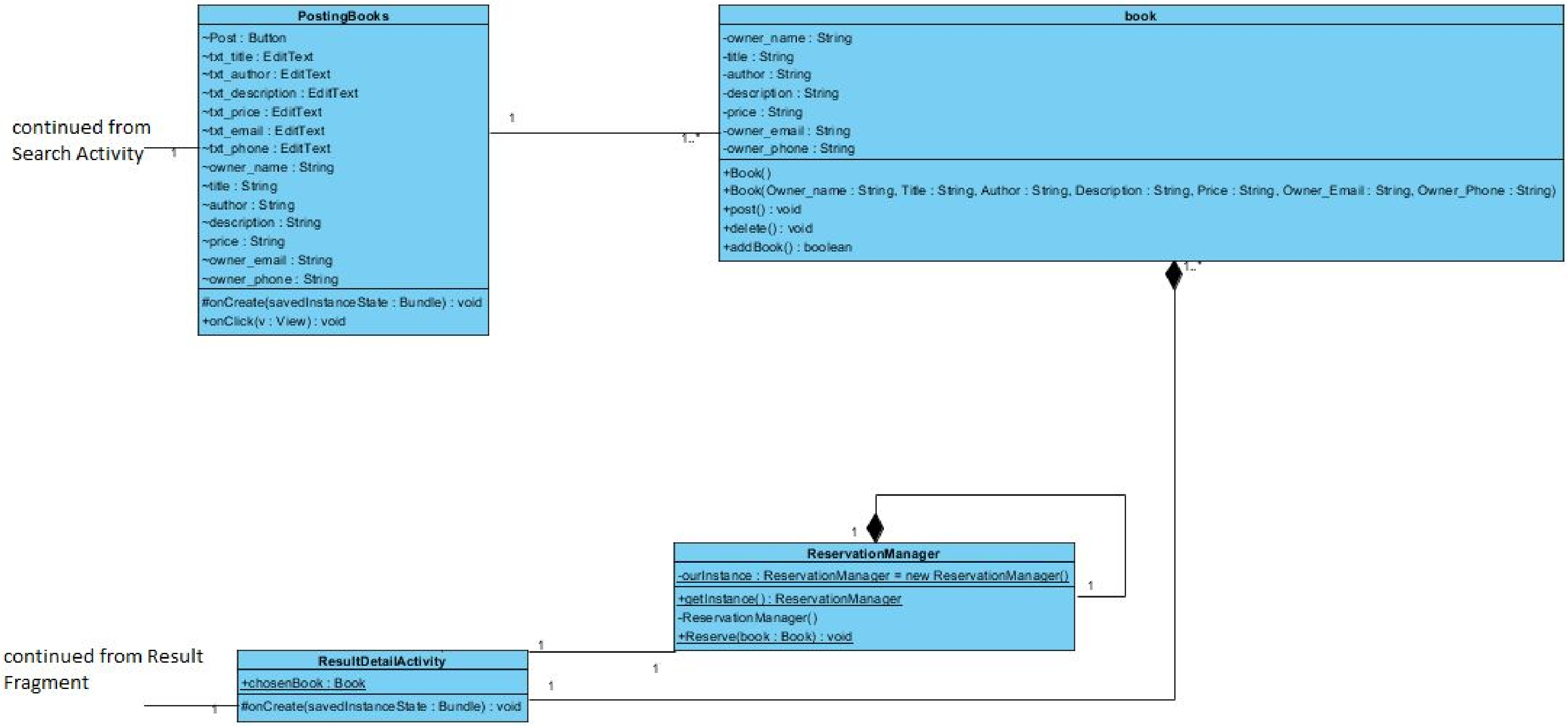
BOOKWISE Class Diagram


MainActivity
~bLogout : Button
~etName : EditText
~etUserName : EditText
#onCreate(savedInstanceState : Bundle) : void
#onStart() : void
-authenticate() : boolean
-displayUserDetails() : void
+onClick(v : View) : void
Login
~bLogin : Button
~bSignUp : Button
~etUserName : EditText
~etPassword : EditText
#onCreate(savedInstanceState : Bundle) : void
+onClick(v : View) : void
SearchActivity
-searchLayout : RelativeLayout
-list : ListView
-price : String
-autoCompleteTextView : AutoCompleteTextView
-arrayAdapter : ArrayAdapter
-b2 : Button
-search : String
#onCreate(savedInstanceState : Bundle) : void
#onResume() : void
+onClick(v : View) : void
+onItemSelected(parent : AdapterView<?>, view : View, position : int, id : long) : void
+onNothingSelected(parent : AdapterView<?>) : void
Register
~bRegister : Button
~etName : EditText
~etUserName : EditText
~etEmail : EditText
~etPassword : EditText
#onCreate(savedInstanceState : Bundle) : void
+onClick(v : View) : void
#validateEmail(email : String) : boolean
#validateUnivEmail(email : String) : boolean
User
~name : String
~username : String
~email : String
~password : String
+User(name : String, username : String, password : String, email : String)
+addUser() : boolean
ThisUser
+name : String
+username : String
+email : String
+reservedBook1 : String
+reservedBook2 : String
+ThisUser(name : String, username : String, email : String, reservedBook1 : String, reservedBook2 : String)
ResultActivity
#onCreate(savedInstanceState : Bundle) : void
Searchmanager
-ourInstance : Searchmanager = new Searchmanager()
+getInstance() : Searchmanager
-Searchmanager()
+searchResult(book : String) : List<Book>
+searchUser(userName : String, password : String) : boolean
+getUser(UserName : String) : void
ResultFragment
-mResultAdapter : ArrayAdapter<String>
-bookList : Book
+ResultFragment()
+onCreate(savedInstanceState : Bundle) : void
+onCreateView(inflater : LayoutInflater, container : ViewGroup, savedInstanceState : Bundle) : View
1
1
1
1
1
1
1
1..*
1
1..*
1
1
1
1
1
1
1
1
1
1
1

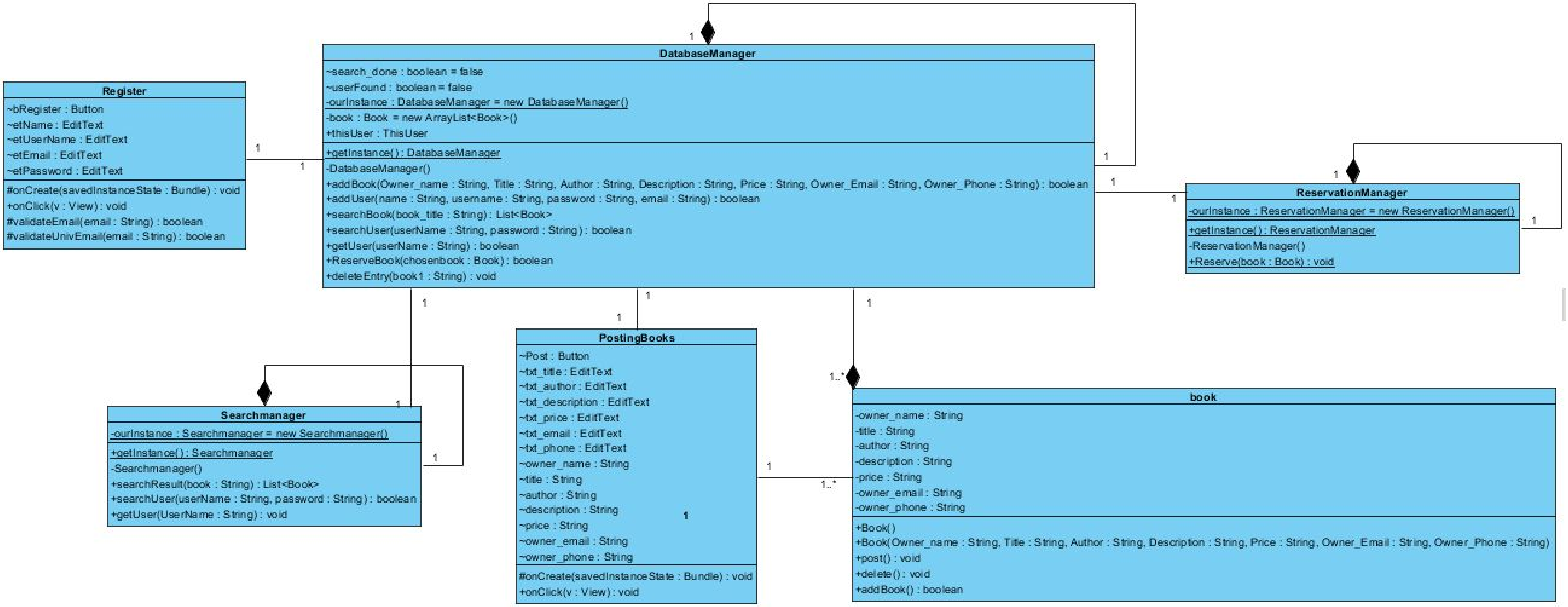
continued to posting books

continued to Result Detail Activity.

BOOKWISE Class Diagram Continued



BOOKWISE Class Diagram Continued


1
DatabaseManager
~search_done : boolean = false
~userFound : boolean = false
-ourInstance : DatabaseManager = new DatabaseManager()
-book : Book = new ArrayList<Book>()
+thisUser : ThisUser
+getInstance() : DatabaseManager
-DatabaseManager()
+addBook(Owner_name : String, Title : String, Author : String, Description : String, Price : String, Owner_Email : String, Owner_Phone : String) : boolean
+addUser(name : String, username : String, password : String, email : String) : boolean
+searchBook(book_title : String) : List<Book>
+searchUser(userName : String, password : String) : boolean
+getUser(userName : String) : boolean
+ReserveBook(chosenbook : Book) : boolean
+deleteEntry(book1 : String) : void
1
1
1
1
Register
~bRegister : Button
~etName : EditText
~etUserName : EditText
~etEmail : EditText
~etPassword : EditText
#onCreate(savedInstanceState : Bundle) : void
+onClick(v : View) : void
#validateEmail(email : String) : boolean
#validateUnivEmail(email : String) : boolean
1
1
ReservationManager
-ourInstance : ReservationManager = new ReservationManager()
+getInstance() : ReservationManager
-ReservationManager()
+Reserve(book : Book) : void
1
1
1
1
1
1
Searchmanager
-ourInstance : Searchmanager = new Searchmanager()
+getInstance() : Searchmanager
-Searchmanager()
+searchResult(book : String) : List<Book>
+searchUser(userName : String, password : String) : boolean
+getUser(UserName : String) : void
1
PostingBooks
~Post : Button
~txt_title : EditText
~txt_author : EditText
~txt_description : EditText
~txt_price : EditText
~txt_email : EditText
~txt_phone : EditText
~owner_name : String
~title : String
~author : String
~description : String
~price : String
~owner_email : String
~owner_phone : String
#onCreate(savedInstanceState : Bundle) : void
+onClick(v : View) : void
1
1
1..*
1..*
book
-owner_name : String
-title : String
-author : String
-description : String
-price : String
-owner_email : String
-owner_phone : String
+Book()
+Book(Owner_name : String, Title : String, Author : String, Description : String, Price : String, Owner_Email : String, Owner_Phone : String)
+post() : void
+delete() : void
+addBook() : boolean

2. Did you make use of any design patterns in the implementation of your final prototype? If so, how? If not, where could you make use of design patterns in your system?

The design patterns that we've implemented:

a) MVP(Model View Presenter)
#Here we are not considering the MVP provided by android F/w. We've implemented MVP in our code as:

(i) CASE-I
MODEL – DatabaseManager
Presenter – SearchManager (It is retrieving the searched book details from the model and updating the view Result Fragment).
View – ResultFragment.

(ii) CASE-II
MODEL – DatabaseManager
Presenter – ReservationManager
View – SearchActivity

b) SINGLETON

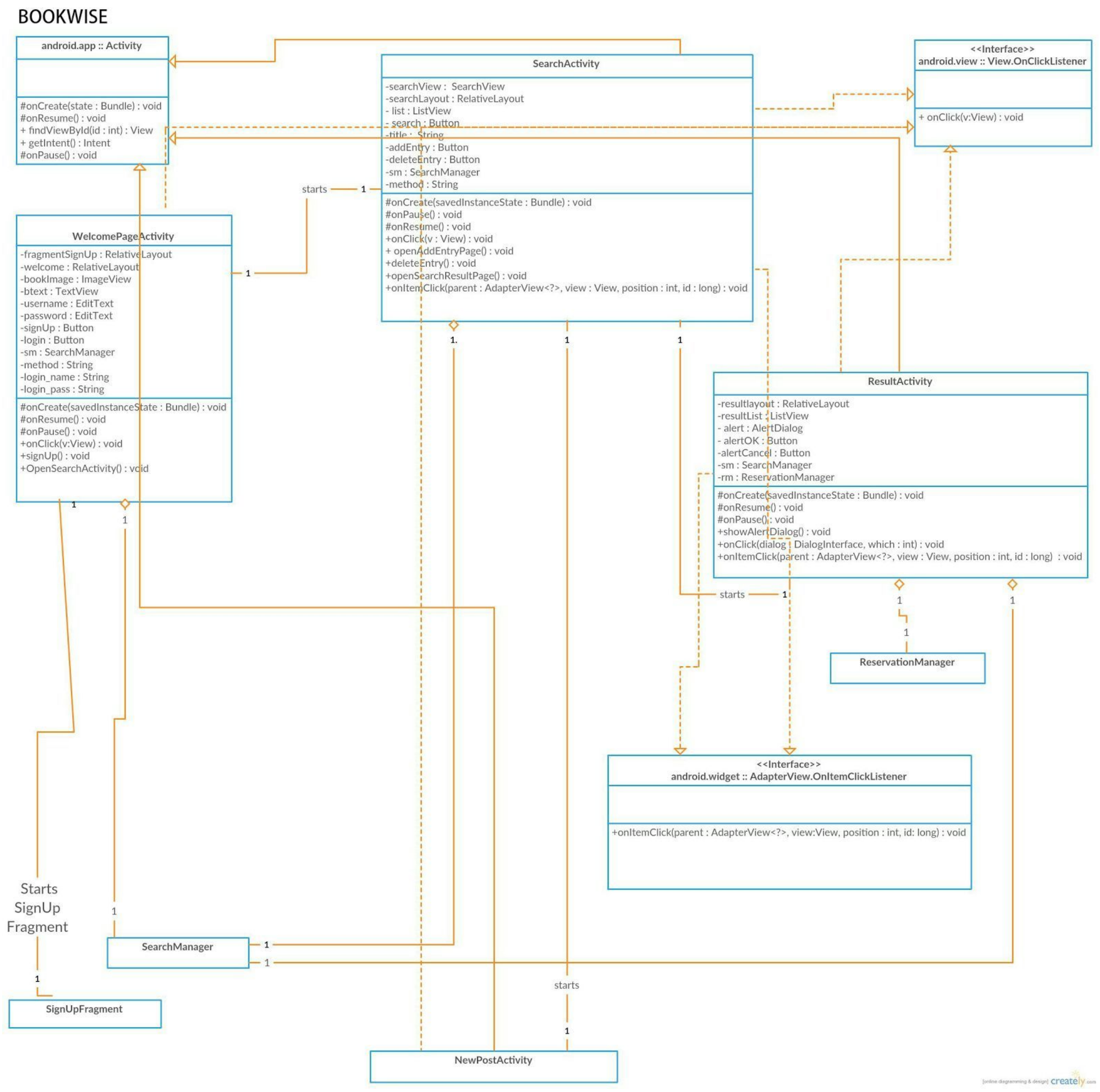
We've implemented Singleton and our singletons are Reservation Manager, Database Manager, Search Manager and have provided a global point of access to them.

The design pattern that we can further use in our app:

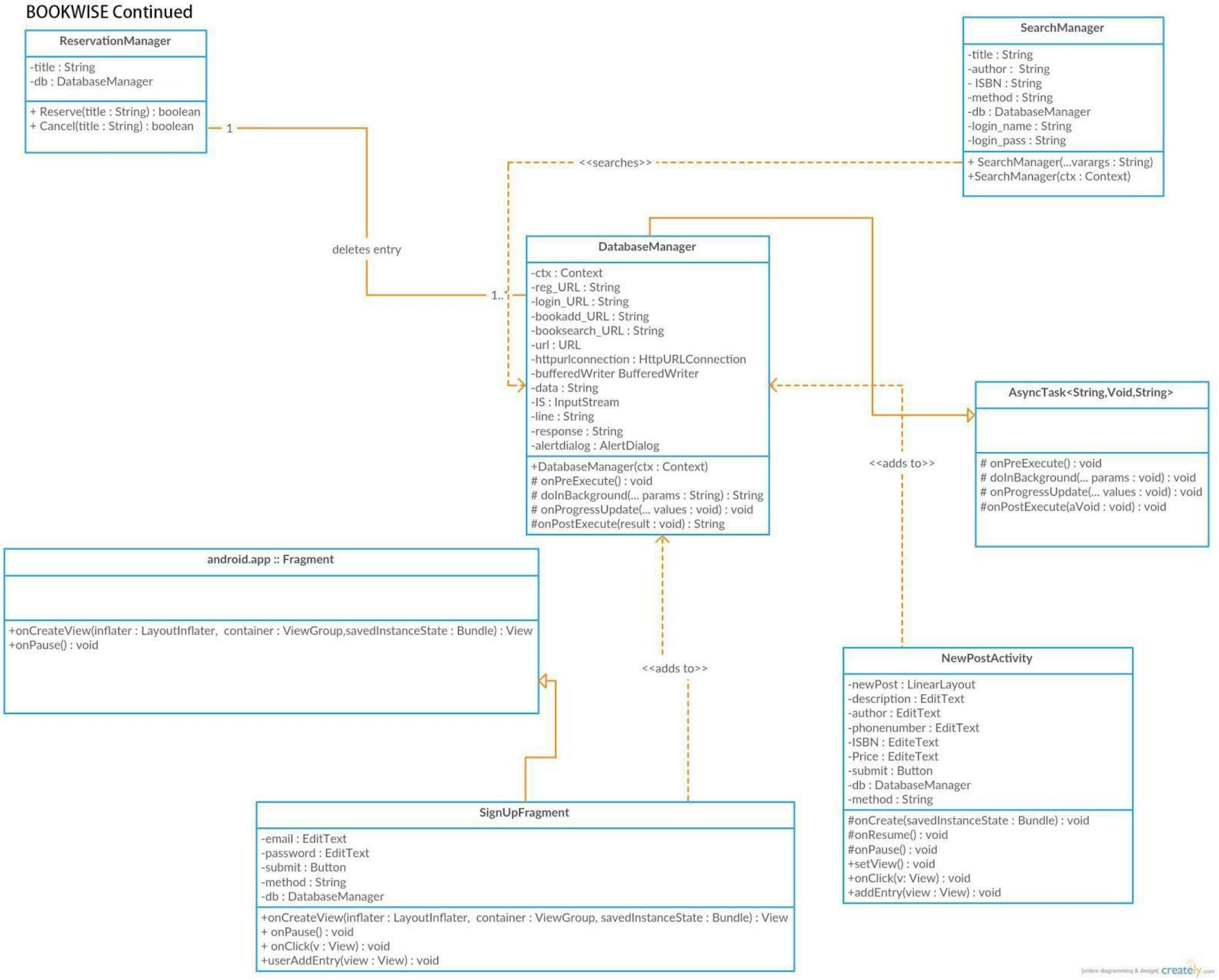
a) Strategy – Say if a user searches a book by using filters like distance within 50 miles, posted in last 6 months etc. For such cases we can implement Strategy design pattern in our app, such that user would get different books based upon his choice of using different filters.

3. In addition, the report must discuss how the final system changed from the design you presented in Project Part 2. In particular, include the class diagram you submitted for Project Part 2 and use it to compare and contrast with the class diagram that represents the final state of the system.

Here is a copy of the original class diagram in part 2.



BOOKWISE Continued



In the development, we mostly followed the design in part 2. However, we did make some changes along the way:

- In the ResultFragment, instead of displaying the listing detail, we just display the all the result returned by the search, to make everything more organized.
- We added a ResultDetailActivity class to display listing detail and deal with the reservation.
- We created User class to achieve signup and login, and ThisUser class to store all the information of the user, and Book class to make reservation and posting book possible.
- Some of the classes names are changed to indicate the function of the class better: WelcomePageActivity - MainActivity, SignUpFragment - Register, NewPostActivity - PostingBook

4. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

We learned a lot from design, analysis and implementing this project. This project is the first and a very valuable experience that we do the design and analysis work before starting coding. It was first challenging because most of us has not used a database and Android before and we had a hard time to figure out the details about our classes and methods. Doing a lot of research before implementation reduced a significant amount of work for coding.

Another thing we learned is that we should have planned ahead since the database part was not as easy as we participated. Implementing database manager class and all the .php files took longer than we thought, so it was pretty stressful in the end. Fortunately, we were able to get the project done on time.

We also learned that if we spend more time on designing, the division of labor in the actual implementation would be much easier. If our App were perfectly designed, each one of us could have just taken a few classes to code and integrate them together. However, during implementation, we had to make a few changes to the class diagram that leads to the fact we need to more communication to get our project done. The bright side of this is that we learned how to communicate with team members more efficiently by helping each other, not hurting their feelings.

The last and the most important thing we learned is that OOP, OOD, and design patterns are not just concepts we have to memorize for interviews. They helped a lot in our development. For example, by taking advantages of OOP and OOD concepts, our App is much more extendable and maintainable. Instead of just selling books, it is very easy to extend our App to sell school-related equipment such as iclickers by using polymorphic reference. The singleton design pattern was also extremely helpful in our case since it strictly managed the classes that are only supposed to instantiate once, such as our searchManager class and databaseManager class.

Overall, this is a great project that all of us learned a lot, not only about programming skills, concepts, and knowledge, but also about design, teamwork, and communication skills.