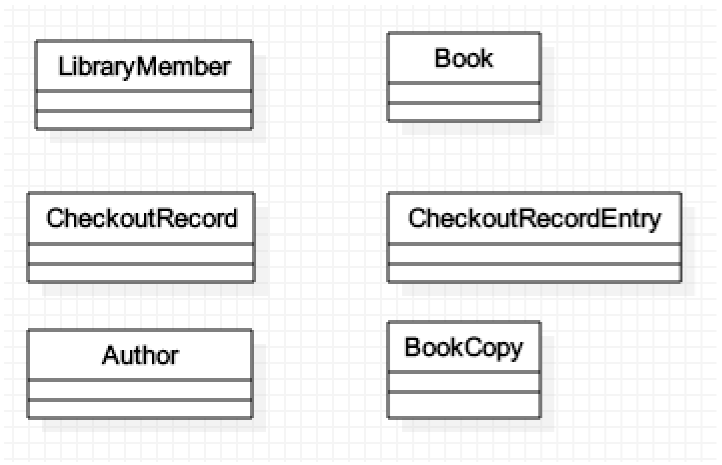
Library System Design Workshop

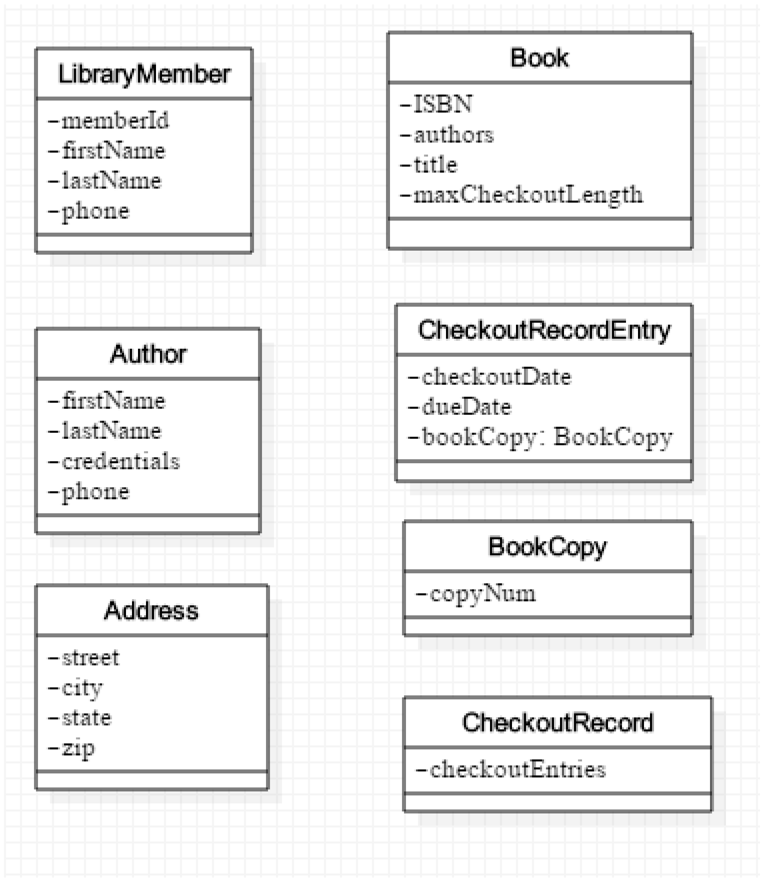
In these notes, we think through many of the steps leading to UML design diagrams for the Library System.

Step 1: Identify Candidate Classes (concepts)

Examine the noun and noun phrases and filter. The following is a reasonable list:



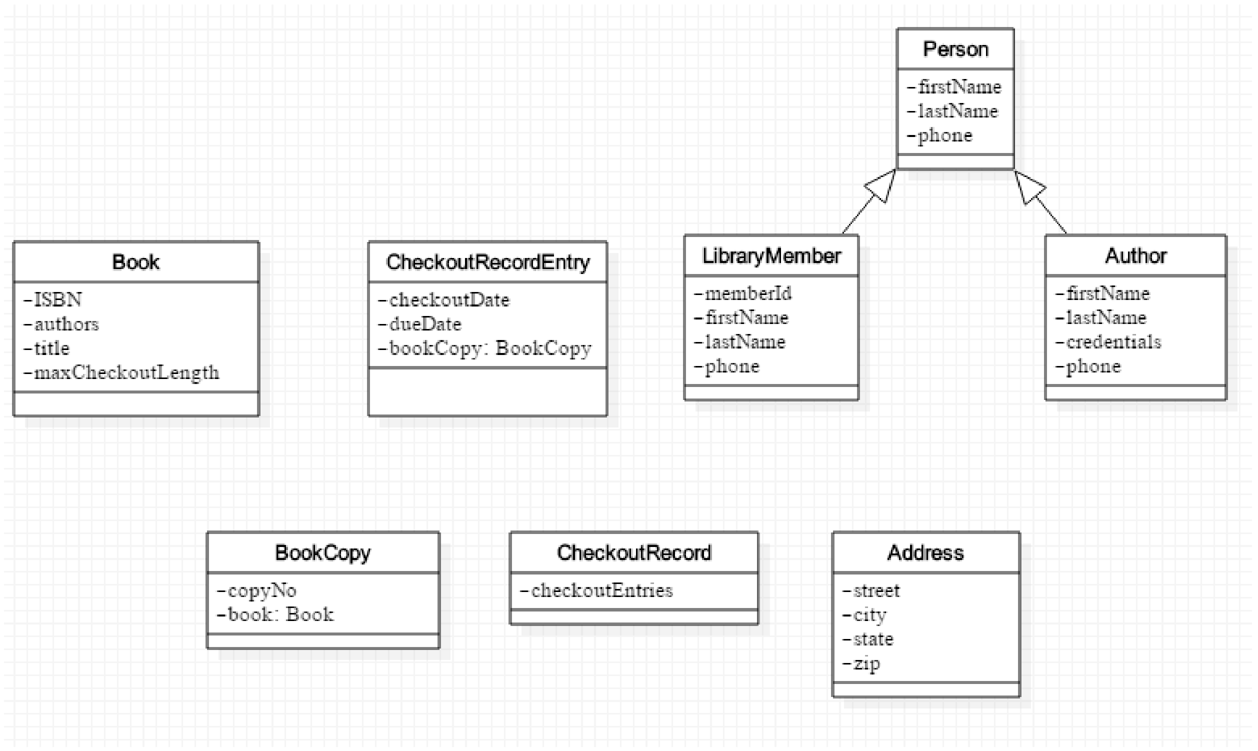
Step 2. Add Attributes



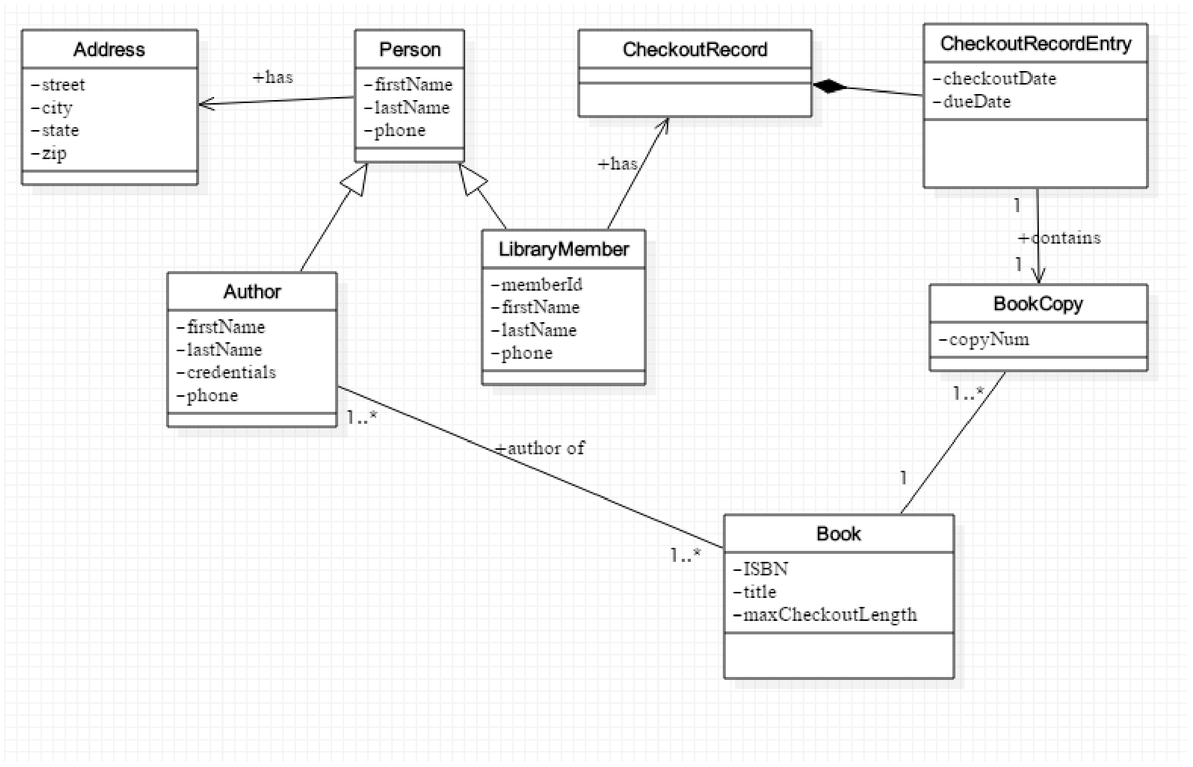
Notes:

1. *Address class*. We have created a separate class for two reasons. First, Address fields are the same for both library members and authors, so we anticipate reuse by factoring out an Address class. Also, it is conceivable that at some point we will want to support a member or author having multiple addresses; this would not be possible to implement cleanly if all address fields were placed individually in the member and author classes.

Step 3. Add Inheritance



Step 4. Add Associations



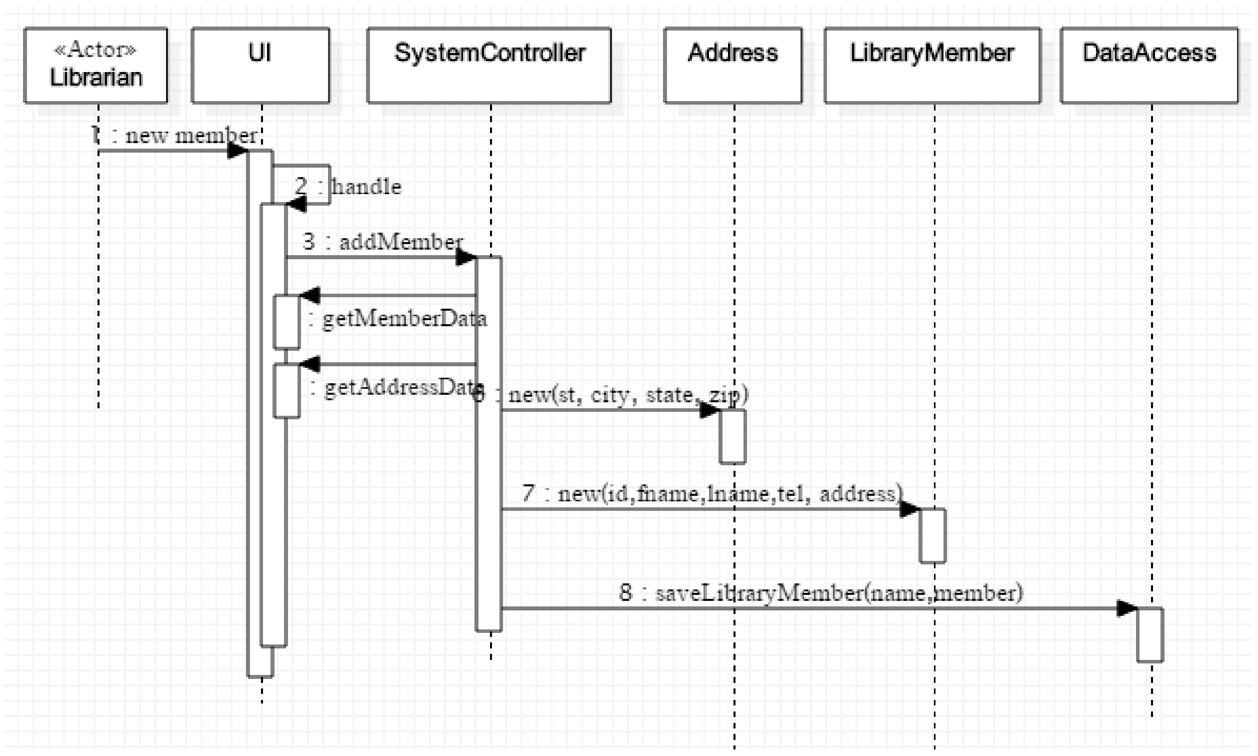
NOTES

1. *Book-BookCopy relationship*. Why not just let numCopies be an attribute in Book? Two reasons: (1) In real life, libraries do keep track of different copies and even label books to indicate the copy number; (2) the problem statement requires us to keep track of a copy number, and this cannot be done if we handle copies only with an attribute numCopies
2. *Association class between LibraryMember and BookCopy?* It is possible to view a CheckoutRecordEntry as information about the "checkout" relationship between a LibraryMember and a BookCopy. We have chosen a different way to model this – when a LibraryMember checks out a book copy, we view it as an interaction between the member and his checkout record – a new entry is added to his record, and that entry includes the book copy that was checked out.

Step 5. Add Operations – Create Sequence Diagrams

One strategy for identifying operations at this stage is to start creating sequence diagrams for the use cases. These will reveal the operations that need to be present to carry out the use cases. Once these are discovered, they can be added to the classes in the class diagram.

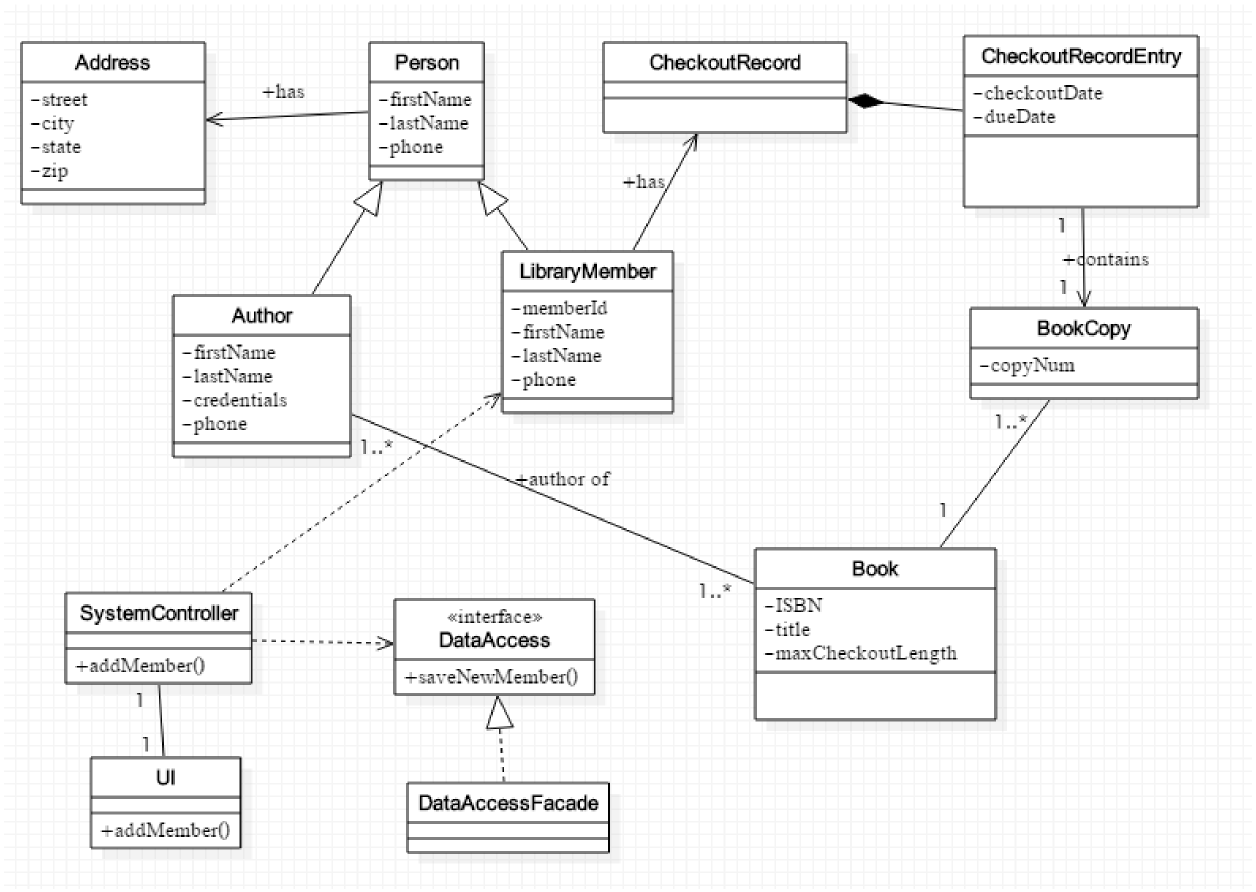
The Add New Member Sequence Diagram



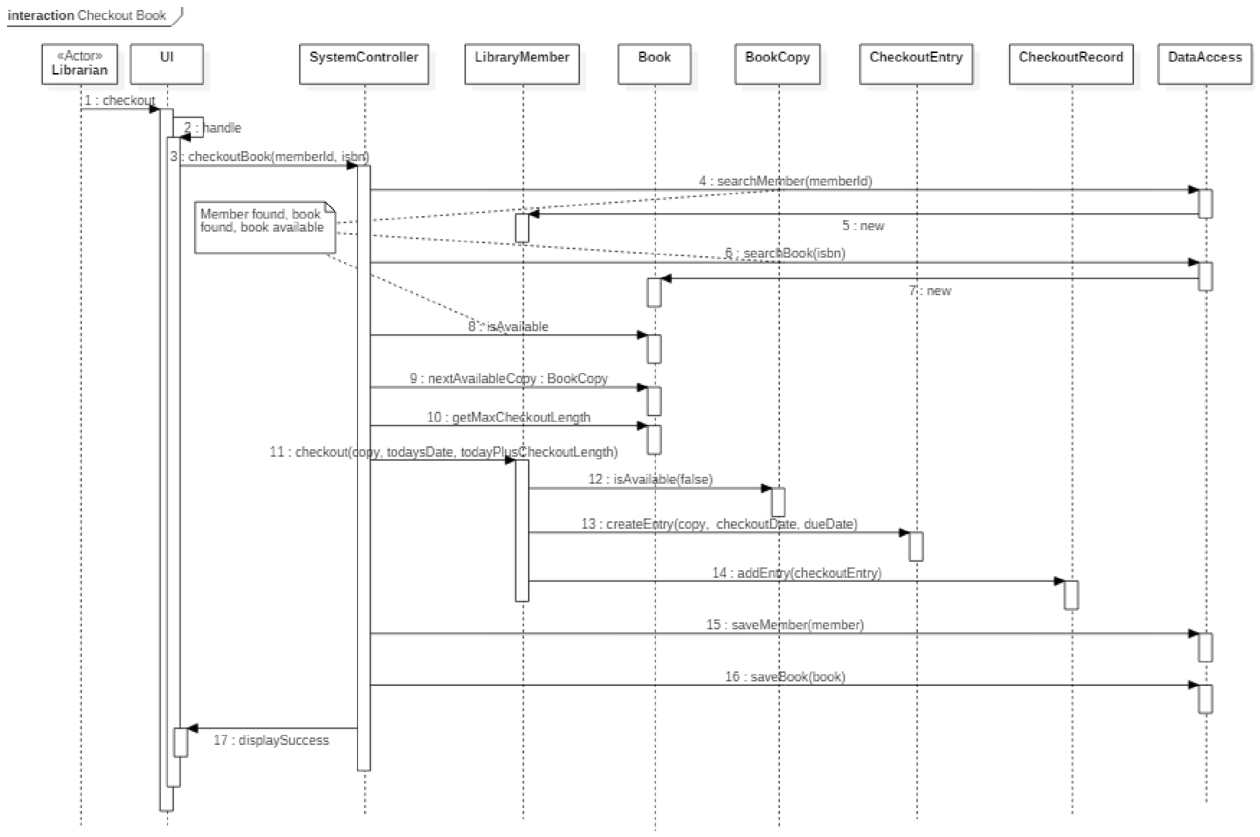
NOTES

1. *Introducing a UI class*. Since it doesn't make sense (by MVC) for the UI to know how to navigate through the deeper layers of the application, we have a controller class that the UI talks to. The controller dispatches requests and returns results.
2. *How should control be distributed for data saves?* Take a look at the `saveLibraryMember` message. Whose responsibility is it to save the member object? It can be argued that Member should be asked to save itself, but it can also be argued that Member is just a bean, just a holder of data, and should not be aware of data sources. The latter point of view is usually handled by embedding Member into a *Data Access Object (DAO)* that knows how to communicate with the data access layer. Here, we let the controller do this communication. In future refinements, DAOs could be added. But it is also fine to ask the Member to save itself too; with this approach, entity classes like LibraryMember could be enhanced to conform to an ORM strategy (like Hibernate or JPA). (this is more like the ORM strategy for frameworks like Hibernate and JPA).
3. What do we learn about operations from this diagram? The diagram reveals 3 new classes and some new operations, which we can add to the class diagram.

Step 6. Add operations for the Add New Library Member use case



Step 7. Sequence diagram for Checkout Book use case (success scenario)


interaction Checkout Book
«Actor» Librarian
UI
SystemController
LibraryMember
Book
BookCopy
CheckoutEntry
CheckoutRecord
DataAccess
1 : checkout
2 : handle
3 : checkoutBook(memberId, isbn)
4 : searchMember(memberId)
Member found, book found, book available
5 : new
6 : searchBook(isbn)
7 : new
8 : isAvailable
9 : nextAvailableCopy : BookCopy
10 : getMaxCheckoutLength
11 : checkout(copy, todaysDate, todayPlusCheckoutLength)
12 : isAvailable(false)
13 : createEntry(copy, checkoutDate, dueDate)
14 : addEntry(checkoutEntry)
15 : saveMember(member)
16 : saveBook(book)
17 : displaySuccess

Notes

1. Many new operations show themselves in this sequence diagram. These should now be added to the class diagram