There are two ways of constructing a software design. One way is to make it so simple that there are obviously no deficiencies. And the other way is to make it so complicated that there are no obvious deficiencies.

—C.A.R. Hoare

Table of Contents

Table of Figures

Question 1

We ask you to conduct a preliminary analysis of a Porterhouse library use case called **Return Books**. Each deliverable is worth 4 marks. **[20 marks]**

You should deliver:

a) An identification of the Actor(s) involved.

Answer:

The actors involved in this use case would be:

i. Librarian
ii. Bar code reader (Assumption: ONLY if the system commissioners thinks to be reading ISBN & Volume IDs using bar code reader instead of manual input)

b) A use case diagram, similar to figure 4.3 of Software construction using Objects.

Answer:

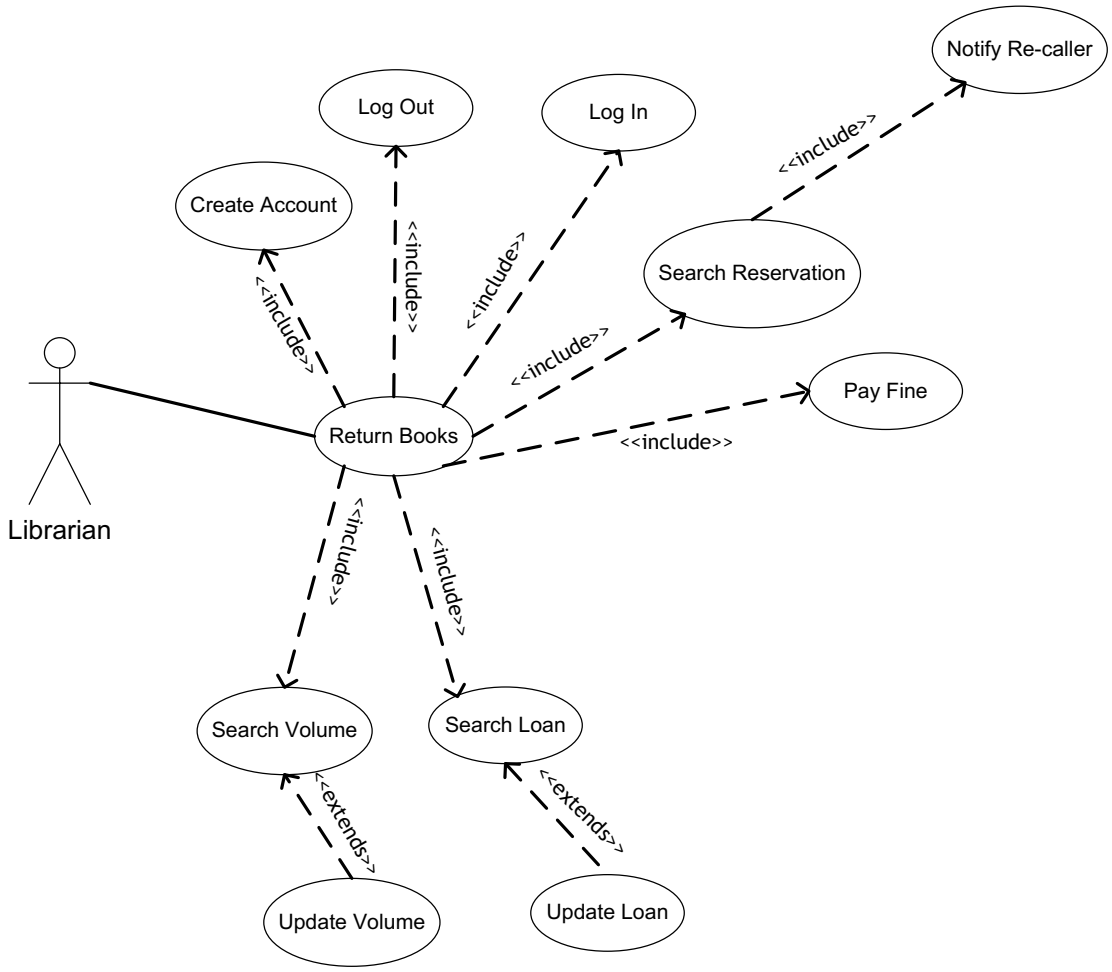

Figure 1: Return Books: Use Case Diagram

I assume that the 'update' use cases embeds search functionality. For example, in order to update a loan record, it is to be searched using the available/suitable parameters. And then the search resultant record details are displayed and available for edit.

The 'login' and 'log out' use cases are used by the librarian to get in and out of the system.

The loan and volume database for the returned copy of book are updated to close loan entry and the book availability status, respectively.

The 'pay fine' use case calculates the fine and updates the user record by adjusting the 'accountValue' attribute according to the fine policy.

The ‘Search reservation’ use case is initiated to check if the returned book was recalled by some other user and if so, then the ‘Notify re-caller’ use case is initiated.

The ‘create account’ use case is used to create a member user of the library. The occurrence of this use case is least possible assuming high security and safe implementation of the system functionality. So, there is a very minor chance that customer records will be compromised.

Note: All these use cases are directly connected to the librarian. In order to avoid the mess in the diagram, I haven’t connected the librarian to all of these using a line as I have used for librarian to connect to ‘Return books’.

Plus, I know using any other notation will risk marks to be deducted but I have no clue then how to show search and update as one use case.

c) A list of possible scenarios similar to that of 4.1- 4.6.

Answer:

1. This use case involves the actor ‘Librarian’.
2. This use case occurs when someone comes to return books and hands them to the member of the library staff (Librarian) or returns books by post anonymously.
3. The **pre-condition** is composed of an individual who is registered with the library and a collection of book titles.
4. The Librarian first logs into the system. (This involves a password.)
5. The Librarian attempts to locate the loans record (If the book is returned anonymously the librarian would search for the loan record using volume identifier of the book.)
6. If no loan record is found the use case terminates.
7. If loan record is found, and the user account involves outstanding fines, the pay fine use case is initiated.
8. The librarian searches the reservations in order to check if the returned volume is subject to recall notices involving other users then the re-caller is notified. The reservations database is updated with the info that user is notified by updating the notify date (that was initially set to be empty).
9. The librarian updates the book availability status.
10. The librarian closes the loan entry.
11. Success state changes include
    (i) Updating the user account by modifying the ‘accountValue’ (fine levied)
    (ii) Updating the volumes database, for books returned are marked by the System as Available/reserved.
    (iii) Updating the loans database by marking the loan entry to be closed and noting the return date.
12. Point 11 represents the **post-conditions** of the main success scenario.
13. Notes:

a) Books are distinguished from volumes in that a given book may be present as several distinct volumes (multiple copies) in the library.
b) One identifies books by ISBN number, but volumes are identified using an internal library code described as volume identifier here (Volume ID).
c) There are categories of users in the library which have correspondingly different returning policies.

This use case description reveals that this analysis implies that other use cases are involved in its execution. We could identify:

Log in
Search Loan
Update Loan
Search Volume
Update Volume
Pay Fine
Search Reservation
Notify Re-caller
Create Account
Log out

Assumption: The notifyDate is important as requirements specify that afters 3 days of notification if the re-caller is not able to collect the books then these are returned to shelf and marked available or issued if someone wants to loan them and the reservation is cancelled.

d) Identification of the main success scenario.

Answer:

Main Success Scenario:

1. Librarian logs into system.
2. Librarian locates the loan record for the volume that is to be returned.
3. Librarian calculates fine.
4. Librarian updates user account for fine if return exceeded the due date.
5. Librarian searches reservations to check if the book was recalled by someone and sends a notification to the user who made the reservation. Reservations database is then updated with the notifyDate.
6. Librarian updates volumes database by updating book availability.
7. Librarian updates loans database by marking the loan entry to as closed.
8. Librarian logs out of system.

e) A detailed use case description similar to the tabular one given in chapter four.

Answer:

Use Case Number: 2	**Return Loan**
Goal	Handle the functionality associated with returning books.
Description	Library user approaches Librarian with a verbal request to return books. Librarian uses the system to locate the account of the user and update it with a list of the volumes returned. No update is made if the account does not exist. If the user owes money from fines then this is also taken care of. If the returned books were recalled by someone else, then reservations are made for it.
Actors **Constraints**	Librarian – a privileged user who may: ▪ update user loan information ▪ create and delete user accounts ▪ make reservation for books ▪ update the record of library volumes Librarian must log in to the system to use it. Librarian is a privileged user and has a login password. This use case must execute in under two minutes with a mean execution time of one minute or less. Librarian should be able to learn the associated activities in under one hour. Screen dialogs should be readable to people with averagely poor eyesight. Screen information should be printable and accessible to members of the public.
Pre-conditions	For the main success scenario these are: ▪ An existing and valid user account showing books loaned ▪ An user allocation that is not exhausted ▪ The library is open for returning books ▪ The library system is up and running
Main Success Scenario	The librarian does successfully return the books by updating the user account with information on the volumes returned. The scenario has these stages. 1. Librarian logs into system. 2. Librarian locates the loan record for the volume that is to be returned. 3. Librarian calculates fine. 4. Librarian updates user account for fine if return exceeded the due date. 5. Librarian searches reservations database to check if the booked was recalled by someone and notifies that user about the

	availability of the book. The reservations database is updated with the notifyDate'. 6. Librarian updates book availability i.e. updates volumes database. 7. Librarian updates loans database by marking the loan entry to as closed. 8. Librarian logs out of system.
Post conditions for main success scenario	▪ The system records access by Librarian. ▪ The volumes database is updated with return information. ▪ The user account is updated with account information. (Adjusting 'accountValue' for fines if any liable.) ▪ The loans database is updated by closing the loan entry. ▪ The reservations database is updated.
Assumptions	The Use Case will allow for one book at a time to be marked as returned. The capability to mark multiple books at one time as returned is not part of this Use Case.
Other scenarios (named in brackets)	1. The user account is found not to be registered with the library. (No user account. Might be deleted or compromised accidentally?) 2. The user account is found to reflect money owed to the library or to have outstanding overdue books. (Account debits / Pay fine.) 3. The library is not currently checking in books. (System Failure/Holiday) 4. Information entered is not in valid format. (Incorrect format) 5. All mandatory fields are not provided. (Missing fields/info)
Related use cases	Use case 5 – Log in Use case 13 – Search Loan Use case 8 – Pay fine Use case 11– Update Loan Use case 7 – Search Reservation Use case 9 – Notify re-caller Use case 14 – Search Volume Use case 10 – Update Volume Use case 12 – Update System Access (automatic logging) Use case 3 – Create Account Use case 6 – Log out All of these use cases involve Librarian.
Frequency of occurrence in the system	The main success scenario is one of the two commonest scenarios in the system (the other being Cancel Return). It represents about 46 per cent of the activities involving Librarian.

Test generation	▪ Each of the eight scenarios described must be tested. ▪ The timing constraint is fairly lax but should be checked. In practice it is unlikely to be an issue in this system but for contractual reasons it may be appropriate to include auxiliary code to collect data on how long issue clerks are taking to record loans, when the alpha system is available, and this may have a bearing on interface design.
Notes	We assume that Librarian has no trouble getting into the system but any password-protected system will occasionally cause problems. We should check on library policy with respect to login problems. The following terms are defined in the system glossary ▪ Librarian ▪ Account ▪ Book ▪ Volume ▪ User ▪ Loan ▪ Reservation

Question 2

Each part is worth 5 marks. **[20 marks]**

a) Construct an *isolated walk -through* for the Return Books use case (that is, construct the walkthrough as if with no other knowledge of the structure of the system). Report your walkthrough in the form of a table similar to the one given in chapter five, (the Borrow Books use case).

Answer:

The walkthrough based on the steps in 'Return Books' use case:

1. Librarian logs into system.

   The librarian enters username and password information in order to get into the system. This step initiates another use case called 'log in'.

2. Librarian locates the loan record for the volume that is to be returned.

   Once the user is located, next the librarian finds the loan record for the book to be returned. This involves use case called 'Search Loan'. Where the loan record is searched using the volume ID, User ID and the issued date of the copy of book.

3. Librarian calculates fine.
4. Librarian updates user account.
   If the returned book copy has exceeded the return date then the fine is to be levied.

   Librarian calculates & adjusts fine using the 'Pay fine' use case and updates the user database by updating the 'account' attribute.

5. Librarian searches reservations database to check if the book was recalled by someone else and notifies that user by email and reservations database is updated by noting the 'notifyDate'.

   This step involves the 'Search Reservation' & 'Notify Re-caller' use cases.

6. Librarian updates book availability i.e. updates volumes database.

Librarian opens the volumes record for this copy of book and marks the book to be available or reserved. This step initiates the use cases 'Search Volume' & 'Update volume'.

7. Librarian updates loans database by marking the loan entry to as closed.

   This step initiates the use case 'Update Loan'.

8. Librarian logs out of system.

   This step uses 'Log out' use case.

b) Using your walk-through, and any other information, suggest candidate classes associated with the Return Books use case (do not include classes that do not occur naturally as part of this restricted analysis). Present arguments based on function and responsibility for these classes.

Answer:

Candidate classes for the Porterhouse Library system:

- Book
- Volume
- Catalogue
- User
- Loan
- Reservation
- Customer

Here I reason the classes stated above.

1. Book:

   This class would be used for catalogue management of the library books etc. We probably would need to update, check and search the library stock, which would be involving querying and searching too. So we would be implementing this with the help of relational database. This class would be corresponding to the book database table by sending messages.

2. Volume
   A book may have several copies and these are identified based on some unique identification system or numbering like labelling them will volume identifiers. A relational database table would be made for volume and this class would interact with the volume database table.

3. User
   This class takes care of the information needed to be stored for the system users or library staff. The permissions or capability attribute of this class will define the user access level or type. This will interact with the User database table.

4. Loan

   When a book is lent to college students or staff, the information regarding loan needs to be saved. This class will handle that by interacting with the loans table.

5. Reservation

   When a book is recalled by a library member, the system user will make a reservation for that when the book is returned and notify the user when the book is available and update reservations database by entering the notifyDate.

6. Customer

   The customers are different from the system users. The customers in this case are the college staff and students. These have different attributes associated with them for instance a student don't need a user name and password as he is not to use the system directly. Similarly, we might not be interested in duplicating the librarian postal address details as we don't need it for the library management. These interact with the customer database as well.

The above are the obvious classes. Now Lets go through the use case step by step, analyzing it.

1. Librarian logs into system.

Step 1 indicates that a class/object is responsible for getting username and password from the login screen, then sends it for verification and if it succeeds then logs the user in to the system.

The User class defined above, stores the information of user name, password and its capability. There needs to be another controller class that would manage all interface links. That is which screen to bring next based on some conditions.

So we need a new class to control the conditional navigation of interfaces.

2. Librarian locates the loan record for the volume that is to be returned.

Here the function of the loan class would be invoked and we have already defined Loan class above.

3. Librarian calculates fine.
4. Librarian updates user account.
(Fine adjusted if return exceeded the due date and marks the book as returned)

This functionality will be handled by the loan class.

The user account that he would update will be of the library member, who is defined as customer above. He might be a student or a staff member. So the customer class also has two 'kind of' users, so two new classes identified.

5. Librarian searches reservations database to check if someone recalled this book. If the book was recalled the re-caller would be notified via e-mail and reservations database updated by noting down the notifyDate.

The search function of the reservation class would be used to find the info from the reservations database. The reservation class has been mentioned above.

The re-caller of the book will be notified and reservations database would be updated. This would be done using the function of the reservation class.

6. Librarian updates book availability i.e. updates volumes database.

The function from volume class would be invoked and handle this updating of book status.

7. Librarian updates loans database by marking the loan entry to as closed.

The loan database would be updated using the function of the loan class. The loan class has already been identified above.

8. Librarian logs out of system.

The log in and log out will be taken care by a class who would be responsible for interface displaying and navigation. This class would be called Admin.

The complete list of classes is as follows:

- Book
- Volume
- Catalogue
- User
- Loan
- Reservation
- Customer
- Student
- Staff
- Librarian
- Manager
- Admin

c) Represent your suggestions in the form of a class diagram, similar to the one given in figure 5.1.

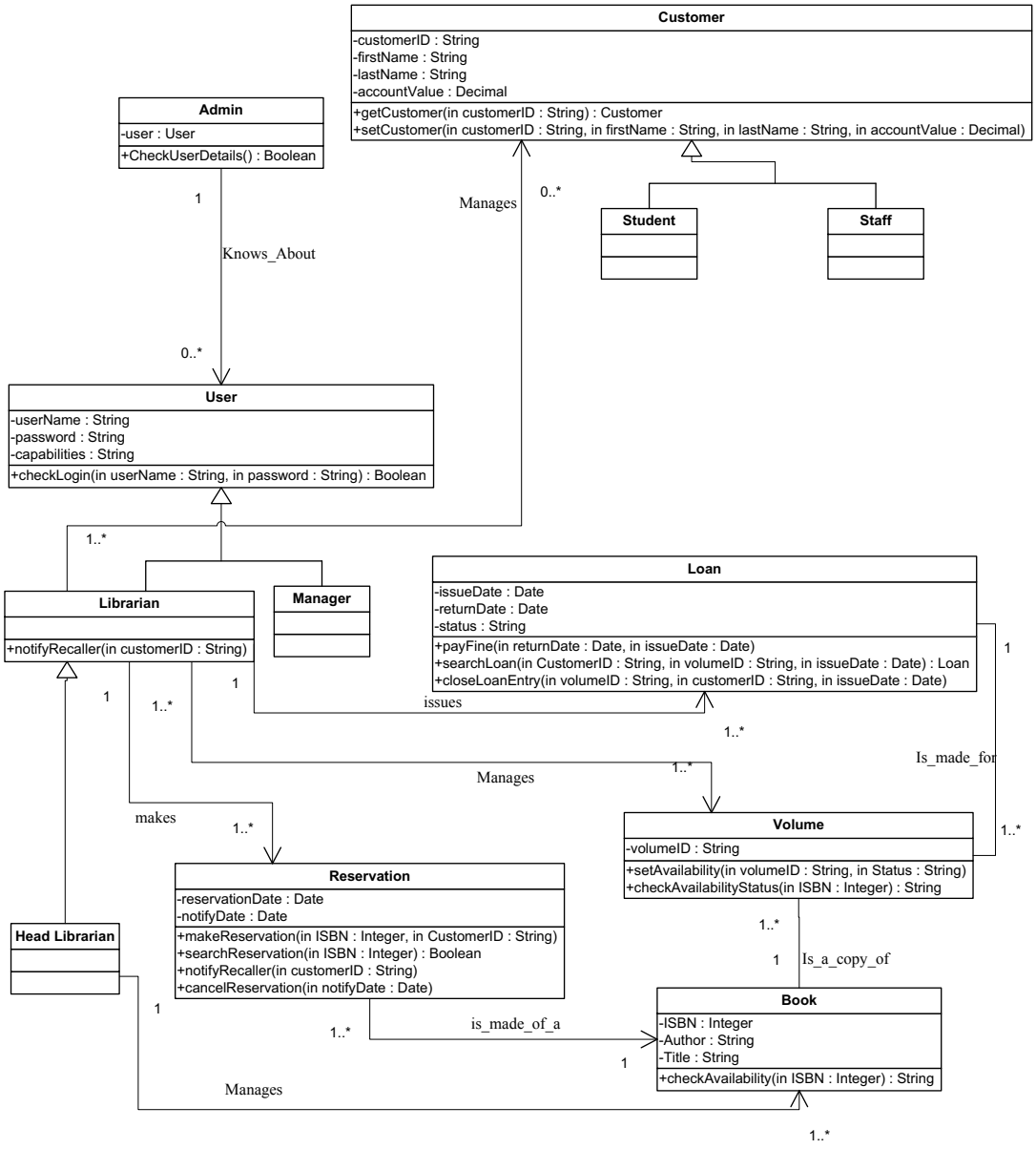

Figure 2: Library System - Class Diagram

d) Construct initial CARC cards for *every* class identified by you in your class diagram. These cards will not be complete, they will simply represent you knowledge based on this initial analysis of a single use case.

Admin <<concrete>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	None	
Subclasses	None	
Description	The Admin object is intended as handling all system dialogues. It controls user access on the basis of passwords and pre-set capabilities by displaying appropriate dialog frames offering particular facilities.	
Responsibility	**Collaborator**	**Association**
1. Knows all the objects in User class. 2. Checks all the proffered user name and password with members of User class	User	Knows_About a 1:* association
Interface Methods	checkUserDetails():Boolean	Called by the system interface in response to user name and password submission
Notes: There will be a single object from this class in the system. It will connect to the user interface and be the channel for user communication with the system.		

User <<abstract>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	None	
Subclasses	1. Librarian 2. Manager	
Description	The User class defines the general attributes and behaviour that its sub classes have in common.	
Responsibility	**Collaborator**	**Association**
1. Informs Admin class if a proffered user name and password exists in the system. 2. Knows about the user name and passwords of the sub class users.		Knows_about a * : 1 association
Interface Methods	checkLogin(username:string , password:string):Boolean	Called by the admin object to validate user

Librarian <<concrete>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	User	
Subclasses	Head Librarian	
Description	The Librarian object is intended to manage all customers, reservations, volumes and loan information. The librarian can add, update and delete the customers. The librarian makes reservations and issues loan. Librarian also updates volume, customer, loan and reservations database.	
Responsibility	**Collaborator**	**Association**
1. Knows about all the Loan class. 2. Knows about all the Reservation class. 3. Knows about all the objects in the Customer class. 4. knows all about the Volume class	Loan Reservation Customer Volume	Issues a 1 : * association makes a 1: * association manages a * : * association manages a * : * association
Interface Methods	checkLogin(username:string , password:string):Boolean	Called by the admin object to validate user

Manager <<concrete>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	User	
Subclasses	None	
Description	The Manager object is intended to manage all system users, access the systems logs and all other privileged tasks.	
Responsibility	**Collaborator**	**Association**
Interface Methods	checkLogin(username:string , password:string):Boolean	Called by the admin object to validate user

According to the given scenario of 'Return books' this class hasn't been analyzed in depth due to it no involvement in it.

Head_Librarian <<concrete>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	Librarian	
Subclasses	None	
Description	The Head Librarian is responsible for stock management of books.	
Responsibility	**Collaborator**	**Association**
1. Knows about all the Book class.	Book	manages a 1 : * association
Interface Methods	checkLogin(username:string , password:string):Boolean	Called by the admin object to validate user

Book <<concrete>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	None	
Subclasses	None	
Description	The Book class is used for the catalogue management.	
Responsibility	**Collaborator**	**Association**
1. Knows about all the Volume class. 2. Responds to the Reservation object with the ISBN number of the available volume object		Is_a_copy_of a 1 : * association Is_made_of a 1 : * association
Interface Methods	checkAvailability(ISBN:integer):String	Called by the reservation object to check the availability of the copy of a book user and the first available volumeID in search is returned

Reservation <<concrete>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	None	
Subclasses	None	
Description	The Reservation class has the responsibility of managing and updating reservation database.	
Responsibility	**Collaborator**	**Association**
1. Knows about all the Book class. 2. Informs the Librarian object if a volume of book is reserved or not	Book	Is_made_of a * : 1 association makes a * : 1 association
Interface Methods	makeReservation(ISBN:integer, customerID:String)	Called by the Librarian object make reservation for a recalled book
	searchReservation(ISBN:interger):Boolean	Called by the Librarian object to check if the book was recalled
	notifyRecaller(customerID:String)	Called to

		notify the re-caller that book is available
	cancelReservation(notifyDate:Date)	Called to cancel reservation if re-caller wasn't able to collect book with 3 days of notified date.

Volume <<concrete>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	None	
Subclasses	None	
Description	The Volume class has the responsibility of managing and updating volumes database.	
Responsibility	**Collaborator**	**Association**
1. Knows about all the Book class. 2. Informs the Librarian object if a volume of book is reserved or not		Is_made_of a * : 1 association manages a * : * association
Interface Methods	setAvailability(volumeID:String, status:String)	Called by the Librarian object to update the availability

		status of volume
	checkAvailabilityStatus(ISBN:interger):String	Called by the Librarian object to check the volume availability and the first volumeID in the search is returned

Loan <<concrete>>	
Date	25/06/08
Version	1
Author	Ayesha Asghar
Superclasses	None
Subclasses	None

Description	The Loan class has the responsibility of managing and updating loan database.	
Responsibility	**Collaborator**	**Association**
1. Knows about all the Volume class. 2. Informs the Librarian object if a volume of book is loaned or not		Is_made_for a 1 : * association issues a * : 1 association
Interface Methods	Payfine(returnDate:Date, issueDate:Date)	Called by the Librarian object to calculate

		fine and update the user account value
	searchLoan(customerID:String, volumeID:String, issueDate:Date):Loan	Called by the Librarian object to get the loan entry
	closeLoanEntry(volumeID:String, customerID:String, issueDate:Date)	Called by the librarian object to close a loan entry.

Customer <<abstract>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	None	
Subclasses	1. Student 2. Staff	
Description	The customer class defines the common behaviour that both its sub classes possess.	
Responsibility	**Collaborator**	**Association**
1. Knows about all the student class. 2. Knows about all the staff class. 3. Informs the Librarian object of		manages a * : *

requested customer info.		association
Interface Methods	getCustomer(customerID:string):Customer	Called by the librarian object to get customer details and a customer object is returned
	setCustomer(customerID:String, firstName:String,lastName:String, accountValue:Decimal)	Called by the librarian object to create a customer.

Student <<concrete>>		
Date	25/06/08	
Version	1	
Author	Ayesha Asghar	
Superclasses	Customer	
Subclasses	None	
Description	The student class deals with all the specialized attributes and behaviour related to the students.	
Responsibility	**Collaborator**	**Association**
Interface Methods		

Staff <<concrete>>	
Date	25/06/08
Version	1
Author	Ayesha Asghar

Superclasses	Customer	
Subclasses	None	
Description	The staff class deals with all the specialized attributes and behaviour related to the staff.	
Responsibility	**Collaborator**	**Association**
Interface Methods		

According to the given scenario of 'Return books' student and staff hasn't been analyzed in depth due to it no involvement in it as customer itself is not reserving the book but the librarian is.

Question 3

Construct sequence diagram(s) relating to the messages associated with **Return Books** use case. You may assume the interface outlined in chapter six and you do not need to worry about library staff logging in to the system. You must be clear about which objects are generating the messages and to what effect. You should present accounts of *all* of the messages appearing in your sequence diagram - including their identifiers, parameters and returns types and giving an informal description of what exactly they achieve. You may find the notes on method specification given in chapter nine useful for this. You do not need to include detailed specifications of helper methods associated with these messages (that is, messages they themselves generate) but their functionality should be described. Be careful not to confuse the *physical* activity of returning books with the software activities associated with it. **[20 marks]**

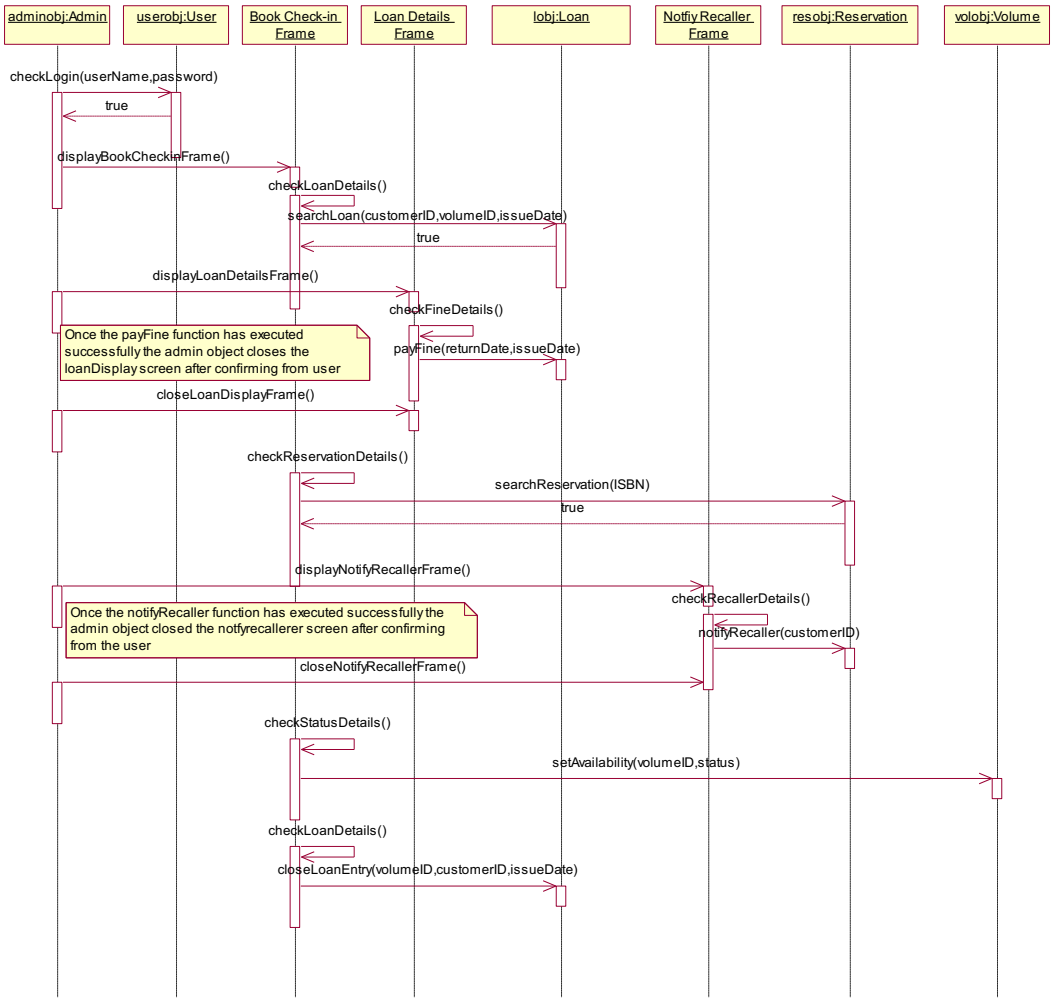

Figure 3: Return Book - Sequence Diagram

(**Note**: I have also attached a jpeg image of the sequence diagram separately in the assignment folder.)

Question 4

This question asks you to review your experience of unit four critically. If you have experience of other methodologies, you may wish to bring that in here, but that is not strictly essential. The assignment is rather open -ended, we are more concerned with the force of your answers than right or wrong solutions. Note that this question is worth 40% of the total mark for the assignment. **[40 marks]**

Software construction using Objects outlines an approach to the analysis and design of systems based on object oriented ideas.

a) **Construct a two hundred word summary of the approach used. Your summary should reference Software construction using Objects. [10 marks]**

Answer

Below are mentioned the steps that the approach used in course material follows:

Identification of use cases

- One can possibly start identifying the use cases by looking at the verbs used in the system specification.
- The activities that the system is supposed to perform are chalked out as use cases.
- The person who carries out the activity also called as an actor is identified.
- The success scenario, possible failure events are also identified.
- Any pre-requisites that are required by the user to perform the activity are also listed.
- Once the activity is performed, what affects it has on the system, these are also recorded.
- Any assumptions, related use cases (the ones this use case would initiate or use), limitations, occurrence and any notes are also written down.
- A use case diagram is also drawn to graphically represent an actor connected to the use case he performs and any related use cases.
- The information is thoroughly reviewed through out in this process in order to clear any ambiguity. At times you might need to rename a use case to better suit it with the function it performs.
- A use case can also be graphically represented by an activity diagram. An activity diagram divides the scenario between the actor and the system and how the flow of events occurs in order to achieve the task that a use case is supposed to perform.

Identification of Classes/Objects

- Next the classes are identified after looking at all the use cases and its relevant information.
- In order to find out any classes that weren't obvious in the use cases, one conducts walkthrough of the use cases. Each step of a use case is to be analyzed and a decision it to be made that which class would be performing them.
- The association, collaboration, responsibilities of the classes are identified. Their attributes and methods are also listed. Once all of this is identified, a class diagram is drawn.
- Next CARC cards are made for each class, in order to show textually its responsibilities, associations, collaborators. The CARC cards are updated through out the process and their versions are maintained. Any super classes, sub classes and interface methods of the class are also noted in them.
- A system glossary is also maintained to define all the terms used in system analysis.

Interfacing

- Wireframes/screens are built for trialling. These are shown to the customer for any feedback.
- In order to better understand the flow of messages between classes and to represent interface design, sequence diagrams are drawn.
- In the process of drawing sequence diagrams for use cases, you might be able to find any missing class association, responsibility or method that you might need.

Database

- Next one needs to decide for data storage either to use object oriented databases or relational databases. The scenario here required relational databases.

Business work flows

- A better understanding of business work flows is gained through drawing state diagrams. The state diagrams help to analyze the different states an object might take in response to messages sent to them. Any missing links can be figured out here.

Testing

- Static testing is used by conducting walkthroughs through the analysis and design artefacts.

b) Suggest a software development which, in your view, would not be best treated using the techniques described in Software construction using Objects. Your answer should:

i. Offer an outline account of the system needed in half a page.

Answer

A local gift shop wants to upgrade its sales and purchase system.

1. Sales Capture

The system would be recording all the sales that will be taking place along with inventory and sales updating i.e. updating the amount of items left of a particular product that is being sold.

- Cashier should be able to enter the product codes manually or be entered via a barcode scanner.
- Description and price of that item will be loaded.
- Quantity or number of item bought should be entered by the user against each item.
- System should calculate current sales total including tax.
- System should reduce inventory quantity when a sale is committed.
- Storage of the entire sale invoices.

2. Payment Authorization

The system will acquire the user of mode of payment i.e. either cash or credit card. It will accept cash as well as credit card payments and for verification of the credit cards it will consult an external system.

3. Handling returns

If a user wants to return some item, then the following two options would be available to him.

i. He can exchange the purchased goods for some other good with the same price and the stocks of the respective goods would be updated accordingly.
ii. On returning the good he can get his money back and the stocks and the total sales would be updated.

4. System Administration

- Login Facility to ensure security.
- Manage all the users using the different aspects of the entire system.

- Assigning and managing product codes for easy reference of product.

5. Handling Stock

Inventory is updated in the following 3 scenarios:

- Inventory will be automatically updated on every transaction.
- Proper inventory updating i.e. incrementing/decrementing number of items of any commodity depending upon supplies from suppliers.
- Updating price of any commodity with respect to the demand and supply.

6. Report Generation

Following reports would be generated

- Daily, weekly and monthly sales transaction reports.
- Inventory reports.

ii. Present a clear argument supporting your view. [10 marks]

Answer

- Changing from a traditional development model to an object-oriented approach is costly and should not be dismissed lightly.
- The costly nature of adopting this Object Oriented paradigm
- This change requires the infamous paradigm shift, meaning you have to completely change your way of thinking and change your business processes as well as invest in training in order to ensure the staff is ready to accommodate the changes. This requires an investment in not only money but time.
- When developing new systems, several developers may be working on a project studying class libraries to find the common objects, or the objects themselves may be in development and not available for use, making it difficult to share the components throughout the new system.
- In using classes and objects in future development efforts, you cannot assume classes are automatically reusable. Instead, inheritance is dependent on a set of rules. It is important to remember, however, that no method eliminates all problems associated with reusability and no method can entirely eliminate all concerns.

- There is some question as the appropriateness of applying object-oriented techniques to legacy software. Object-oriented methodology starts with the object and this object drives the definition of further needs. With legacy system, the data and its associated model already exist. You also have analysts steeped in traditional methodology more comfortable building the data model first. The good news is that once an analyst uses an object-oriented approach he is resistant to going back to traditional methods.
- In order to adopt this methodology, the business should invest in training for all individuals involved in the analysis process and slowly integrate the process by using it in low-risk projects first to allow the analysts to learn from their mistakes. Successful adoption of this methodology lies in commitment, training, and experience.
- A study by Potok et al has shown no significant difference in productivity between OOP and procedural approaches.
- OOP technology has generated more confusion than almost any other computer technology. For example, many OOP "experts" claim that most companies are either not using OOP properly, or are not taking advantage of OOP. Experts also say that there is a long learning curve before many people grasp the power of OOP; that its benefits can't really be taught, at least not understood, from a book. It has almost become like Quantum Physics, in which only a small elite group appears to understand it properly, and everybody else needs years of meditation and practice.
- Persistent storage (databases) is especially in turmoil in the OOP world. Some experts say that a company cannot get the benefits of OOP without using Object-Oriented Database systems; others say that companies need to hire "middle-layer" experts to convert conventional databases into virtual OO databases for other programmers. However, there are very few books on database connections to OOP (relative to other OOP titles), and probably fewer experts.
- OOP generally does not map well to relational databases (RDBMS). Using relational tables with OOP often requires converting (mapping) fields into objects and visa-verse when putting them back into the tables. This is a painstaking process and can waste a lot of programming time.
- Data in traditional relational form is relatively easy to transfer to different systems as technology and vendors change. Procedures are not easy to transfer. If you mix the data in with methods, then you are **stuck** with the current OO programming language to interpret your data. You cannot easily read the data except with the OOP language and/or program that generated it.
- One of the biggest differences between OO and procedural/relational is that OO tends to group operations around nouns (entities), while procedural tends to group around tasks/activities (at least the code

part). I tend to find task grouping more "invariant" (stable) than noun grouping.

- OO is more suitable for ecommerce applications or applications involving complex data types.
- OO does not support component based development.

c) Suggest *two* potential weaknesses to the Software construction using Objects approach to analysis and design - illustrating your answer with examples based on the library system described. Write no more than 700 words. (You may include diagrams, we expect you to refer to course material, and you may also refer to your answers in the first two parts of this question). [20 marks]

Answer:

Object methodology has many benefits but at the same time it is not without its failings. Some of its weaknesses are being addressed but at a rather slow pace.

The weaknesses in object methodology are not mainly technical but its slow absorption by users in applying the methodology into existing systems development.

Some of the weaknesses are:-

1. **Lack of system decomposition** – Decomposition is important because many systems are too large to be developed by one team in order to meet the deadline. Therefore, systems need to be broken down into smaller components that can be assigned to multiple teams to be worked on concurrently. Decomposition needs to be initiated at an early stage of the development process. Decomposition must have a strong connection between components to allow smooth integration of the components at a later stage. In OO methodologies there are difficulties in breaking systems down into smaller components.

   There is a possibility that the college is in a hurry to get the system done or to put it in a better way, saving time though incurring costs. The components of the systems can be made and developed in parallel. Structured programming supports components based programming and is best suited for this situation.

   In contrast to component-based systems, object-oriented systems are arranged, for the most part, in class hierarchies. The nature of this arrangement means that classes at lower levels inherit characteristics and processes from the parent classes above them. This makes it difficult to reuse these classes in different

environments or combinations since reuse depends first on inheritance, and secondarily on encapsulation. It also tends to curb information hiding, since object users must understand how characteristics and processes inherited from parent classes are implemented in child classes.

Since components are independent from one another, there are no inheritance concerns. Encapsulation hides the way a component's behaviour is implemented so clients needn't know the details behind a behaviour's implementation. Ideally, components that share the same interface should be interchangeable; a single component with a well-defined interface can plug into multiple environments. Therefore, component reusability depends almost entirely on encapsulation.

CBD (Component based development) provides a number of advantages to business. Components bring unprecedented flexibility to application development as they can be added, removed, or changed without disruption to the system.

This is especially useful in multi-tier applications where business processes may change faster than the application. Components cut the need for extensive, time-consuming upgrades. They offer a presentation channel, back-end application, and database independence that allow developers to look at or work with single components without disturbing the entire system, allowing a more rapid response to changes in business.

The methods for designing CBD solutions help an organization maintain its focus on the major pieces of its domain, and the interaction of those pieces.
A component management and assembly infrastructure can knit together all the pieces specified, built and acquired, even when each has been developed using different people or different technologies.

Efficiency also increases through the reuse of component code and technical standards. Additional benefits accrue as components give developers the freedom to concentrate on business challenges rather than having to contend with low-level middle-ware services. CBD can even make a competitive advantage of a company's application delivery, since faster development and easier maintenance of existing functionality are built-in benefits.

Components end the struggle to design, develop, test, and debug systems. This, of course, shortens overall development time and reduces cost. It's an optimal time for companies to consider CBD as an alternative to Object-Oriented Application Development (OOAD) since it has evolved sufficiently from OOAD, and proven its functionality.

The loan and return of books can be the two major components of the system, which are to be developed in parallel and isolation but using OO approach this would not be possible. As in OO approach all the objects form one system in

which they are related and connected. OO makes component development difficult.

2. **Lack of end-to-end process modelling** – According to Fichman and Kemerer, global processes, which involve forward and backward execution of intermediate steps between start and end, exist in many problem domains. Although OO methodologies use operations to model parts of the process, there is no specific model to describe global processes. Therefore, a separate tool is required to arrange different encapsulated operations into a model that describes global processes.

3. **Lack of supporting programming languages /programming skills in OO** – Although OO methodologies can be implemented using traditional programming languages, special languages supporting OO features are needed to facilitate the implementation of the OO systems. OO programming languages must have the ability to create classes, objects, subclasses {through inheritance), and to support messaging and dynamic binding. Only recently that programming languages such as JAVA and C++ are beginning to be widely used by organizations especially with the proliferation of the Internet. Many organizations cannot benefit from the use of OO methodologies until there are sufficient trained programmers who are skilful in OO programming and methodology.

   As mentioned in answer of part (ii), OO requires skilled resources.

   Changing from a traditional development model to an object-oriented approach is costly and should not be dismissed lightly. Therefore, the nature of adopting this Object Oriented paradigm is costly.

   This change requires the infamous paradigm shift, meaning you have to completely change your way of thinking and change your business processes as well as invest in training in order to ensure the staff is ready to accommodate the changes. This requires an investment in not only money but time.

4. **Lack of supporting databases and tools** – OO databases are required to store objects permanently on files. Currently, there are not many data base management systems that can handle applications such as CAD/CAM, which require many complex objects to be created and stored. Quality OO tools is needed to automate the analysis and design processes. Currently there are some OODBMS such as GEMSTORE, O2, JASMINE, etc., but user awareness or extensive usage is still limited.

   Most OODBs suffer from the lack of query facilities. In those few systems that provide significant query facilities, the query language is not ANSI SQL compatible. The query facilities do not include nested sub-queries, set queries

(union, intersection, difference), aggregation functions and GROUP BY, or joins of multiple classes –facilities fully supported in the RDBs.

While RDBs support authorization, most OODBs do not support authorization. RDBs allow users to grant and revoke privileges to read or change the definitions and tuples in relations and views

Some OODBs require users to explicitly set and release locks. RDBs automatically set and release locks in user processing query and update statements.

In an RDBMS modifying the database schema either by creating, updating or deleting tables is typically independent of the actual application. In an OODBMS based application modifying the schema by creating, updating or modifying a persistent class typically means that changes have to be made to the other classes in the application that interact with instances of that class. This typically means that all schema changes in an OODBMS will involve a system wide recompile. Also updating all the instance objects within the database can take an extended period of time depending on the size of the database.

An OODBMS is typically tied to a specific language via a specific API. This means that data in an OODBMS is typically only accessible from a specific language using a specific API, which is typically not the case with an RDBMS.

As in the library case example, we decided to a relational database due to the factors that OODB requires training costs. Plus, it doesn't support a lot of features that are mentioned above, where as relational databases provide these facilities and it is being used by many programmers for several years.

5. **Lack of reusable software** – Software reusability claimed by OO methodologies may be difficult to implement and achieve. Pre-defined objects need to be well catalogued, documented, and easy to understand to facilitate their reuse. DCOM and CORBA are two examples of OO software architecture that addresses reusable component issues, but again the application is not widespread. Companies have to make a strong commitment to change to these new methodologies in order for them to be implemented. Investment at the front end must be made in order to reap the benefits from this new approach. Software components must be carefully designed to allow future reusability. The whole organization must be involved in identifying and designing organization-wide objects which help to establish the basis for stable, long term systems. Increased training costs are required for this approach.

References

[Booch, 1991]
Booch, G. (1991). Object-Oriented Design: With Applications. Redwood City, CA, USA: Benjamin/Cummings.

[Conger, 1994]
Sue Conger, *The New Software Engineering,*ITP 1994, pp. 39-40.

[Fichman & Kemerer, 1992]
Fichman, R.G. and Kemerer, C.F., (1992) "Object-Oriented and Conventional Analysis and Design Methodologies: Comparison and Critique," ***IEEE Computer*****,** 25(10), 22-39.

[Ghezi et al, 1991]
C. Ghezzi, M. Jazayeri and D. Mandrioli,*Fundamentals of Software Engineering*, Prentice Hall, 1991.

[Potok et al, 1999]
Thomas E. Potok, Mladen Vouk, and Andy Rindos. "*Productivity Analysis of Object-Oriented Software Developed in a Commercial Environment.*" Software – Practice and Experience, Vol. 29, No. 10, pp 833-847, 1999. Available at world wide web at: http://www.csm.ornl.gov/~v8q/Homepage/Papers%20Old/spetep-%20printable.pdf

[Pressman, 1997]
Roger S. Pressman, "*Software Engineering – A Practitioner's Approach*", McGraw Hill 1997, p. 278.

[Schach, 1996]
Schach*, Classical and Object-Oriented Software Engineering*, 3rd. Ed., IRWIN, 1996.

[Sommerville, 1992]
I. Sommerville*, Software Engineering*, Addison- Wesley, 1992.