Object Oriented Design

Design Constraints

- When passing from analysis to design, it is necessary to consider constraints that may exist.

  These include:
  - *Target Environment* - the hardware and software under which the application will run.
    - Is it important to isolate hardware/operating systems dependencies to facilitate porting to a different environment?
  - *Language* - what programming language will be used to implement the application? While it is desirable to use an OO language such as C++, a procedural language could be used.

Design Constraints (cont’d)

- *Supporting Software Components* - what packages or libraries will be utilised. These include
  - User Interface packages such as MFC, OWL, Motif etc.
  - Database Management - such as Oracle RDBMS or some OODBMS.
  - Class Libraries for Container classes, strings etc.
- *Performance Requirements* -Are performance requirements so tight that it may be necessary to sacrifice design elegance for speed.
- *Others*
  - Is memory likely to be severely limited.
  - What are the requirements on authenticating users of the system?

Enhancing CRC Syntax

- In the design stage, several enhancements can be made to the information on CRC Cards.
  These include:
  - Sub-responsibilities:
    - To perform a responsibility, it may be appropriate to break this up into steps.
  - Collaborating Responsibility
    - As well as showing the collaborator(s), it is useful to show which of their responsibilities will be called for.
  - Data Passed
    - The data sent to a collaborator to enable the fulfilling of a responsibility can also be shown

Sub-responsibilities

- To illustrate the recording of sub-responsibilities, the “check in Lendable” and “check out Lendable” responsibilities of the Librarian can be broken into discrete steps

Librarian	
check out Lendable	Borrower, Lendable
check in Lendable	Borrower, Lendable
search for Lendable	Collection
display message	*UI Subsystem*
get info from user	*UI Subsystem*

Librarian	
check out Lendable	
check Borrowers status	Borrower
check it out	Lendable
update Borrower	Borrower
check in Lendable	
check if overdue	Lendable
check it in	Lendable
update Borrower	Borrower
record fine	Borrower
search for Lendable	Collection
display message	*UI Subsystem*
get info from user	*UI Subsystem*

Collaborating Responsibility

- The Librarian card can be further enhanced to show which responsibilities of the Borrower and Lendable will be involved:

Librarian	
check out Lendable	
check Borrowers status	Borrower :can borrow
check it out	Lendable : check out
update Borrower	Borrower :know Lend
check in Lendable	
check if overdue	Lendable : know if overdue
check it in	Lendable : check in
update Borrower	Borrower : know Lend.
record fine	Borrower : know fine
search for Lendable	Collection : retrieve
display message	*UI Subsystem*
get info from user	*UI Subsystem*

Collaborating Responsibility (cont'd)

and also the data that will be passed:

Librarian	
check out Lendable	
check Borrowers status	Borrower :can borrow
check it out	Lendable : check out(Borr)
update Borrower	Borrower :know Lend(Lend).
check in Lendable	
check if overdue	Lendable : know if overdue
check it in	Lendable : check in
update Borrower	Borrower : know Lend.(Lend)
record fine	Borrower : know fine(fine)
search for Lendable	Collection : retrieve(info)
display message	*UI Subsystem*
get info from user	*UI Subsystem*

Interface Classes

- It was decided that, although Unix is being used now, a move to Windows or Macintosh is likely soon. Therefore:
  - Introduce a "User Interacter" class to get information from, and send messages to, the user.
  - This provides an interface to whatever user interface system is chosen.
- Also, it is at present uncertain what Data Base Management system will be used, so:
  - Introduce a "DB" class to interact with whatever DBMS is chosen.

Object Lifetimes

Certain questions are important regarding objects of each Class in the application:

- What is the lifetime of the object?
- Who is responsible for creating/destroying the object?
- What happens when it is created and destroyed?

Object Lifetimes – Persistent objects

- The lifetimes of some objects in the Library system, and what they must do on start-up and day's end, are reasonably clear:
  - DB
    - Created on start-up, when Data Base must be opened.
    - Destroyed at shut-down when Data Base closed.
  - Librarian
    - Created and destroyed at start-up and shut-down.
    - Need to add responsibility "wait for user".
  - User Interacter
    - Created and destroyed at start-up and shut-down.
    - Need to add "get employee ID" as responsibility.
      - The need to "get employee ID" raises the problem of verifying this ID. Best to put info about valid ID's with one object - "ID Verifier"

Object Lifetimes – Transient objects

With Lendable and its sub-classes and with Borrower:

- The information represented by these objects has a life much longer than any one run of the Library application.
- The number of things represented is very much greater than are likely to be referenced on any one day.

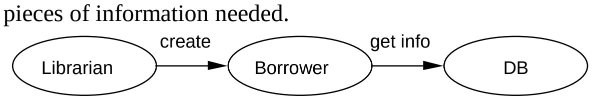
The lifetime of these objects will not exceed the time of interaction with one user.

- Once a User has been verified, a Borrower object must be constructed from information in the Data Base relating to this User ID.
- When the User leaves, the Data Base must be updated and the Borrower destroyed.
- During this period, several Lendable objects may also be created from Data Base information and then destroyed. If necessary the data base is updated before destruction.

When for example the Librarian wishes a Borrower object to be created/destroyed, who should have the responsibility?

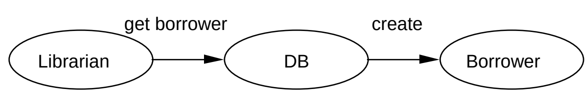
Object Creation/Destruction

- One possibility is for the Borrower class to have create/destroy responsibilities.
  - The constructor could be passed a valid User ID and the ask the DB object for the pieces of information needed.



  - Similarly, the destructor could send to the DB the information that needs to be saved.

- However, at times (e.g. when passed as parameters) temporary objects are created and destroyed without explicit programmer command.
  Better to have DB (which exists to provide interface between application and Data Base) accept responsibility for creating and destroying Borrower (and Lendable) objects.



Check-Out Scenario (1)

What happens when Barbara Stewart, who has one outstanding book, not overdue, checks out a book entitled *Effective C++ Strategies*?"

- *Librarian*: collaborates with User Interacter to "get employee ID".
- *Librarian*: collaborates with ID Verifier to "verify employee ID".
- *ID Verifier* : collaborates with DBMS to verify.
- *Librarian*: passes employee ID to DB to get a Borrower. Sets this as "current borrower".
- *DB*: using "get borrower" responsibility, "get data from DBMS" and collaborate with Borrower to "create borrower".
- *Borrower*: during creation needs a set of previously borrowed Lendables. Asks DB for these.
- *DB*: has set of IDs for the Borrower's Lendables. Creates a Book object to give to Borrower to add to set.

Check-Out Scenario (2)

- *Borrower*: uses "know set of lendables" to add book to set.
  - Decides to implement set of Lendables via a List class.
  - Breaks down "know set of lendables" into:
    - "add lendable to set"
    - "delete lendable from set"
    - "retrieve lendable from set"
    - "number of lendables"
- *Librarian*: uses User Interacter to get choice from user. Choice is "check out". Uses Borrower to "check user status".

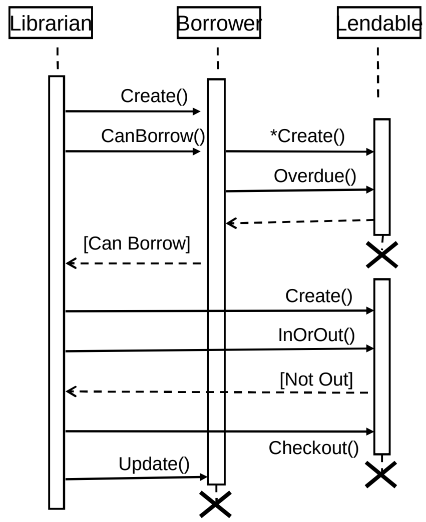
Check-Out Scenario (3)

- *Borrower*: uses “know fine amount” to see that total fine is less than $100. Uses “number of lendables” to see that has only one out. Then must “retrieve lendable from set” and collaborate with Book to see if overdue.
- *Book*: has “know if overdue” as responsibility. Should this be stored or worked out each time? Work it out. Send Due Date to Date
- *Date*: needs to overload operator<(), and can then compare Due Date with Today’s Date. Where is Today’s Date? Add responsibility “know today’s date”.
- *Book*: reports back to Borrower that not overdue.
- *Borrower*: reports to Librarian that can borrow.

Check-Out Scenario (4)

- *Librarian*: ask User Interacter for lendable ID.
- *User Interacter*: adds “get lendable ID” to responsibilities.
- *Librarian*: asks DB to get Lendable.
- *Librarian*: asks Book to check itself out.
- *Book*: set Borrower, set status to OUT. Gets Date to calculate Due Date.
- *Date*: adds responsibility “add days to date”.
- *Librarian*: asks Borrower to add Book to list of lendables. Asks DB to store Book.
- *DB*: stores Book via “put lendable”.
- *Librarian*: ask User Interacter for next request. Gets “exit”. Gives back Borrower to DB.
- *DB*: stores Borrower via “put borrower”.

Check Out Sequence Diagram



Check-In Scenario (1)

- *Librarian*: gets "current Borrower" as for Check-out. Gets "check in" from User Interacter as user's choice, and also Lendable ID.

  Collaborates with Borrower this time to get Lendable.
- *Borrower*: uses "retrieve lendable from set" to get Lendable.
- *Librarian*: could ask Book if it is overdue and what fine amount is.

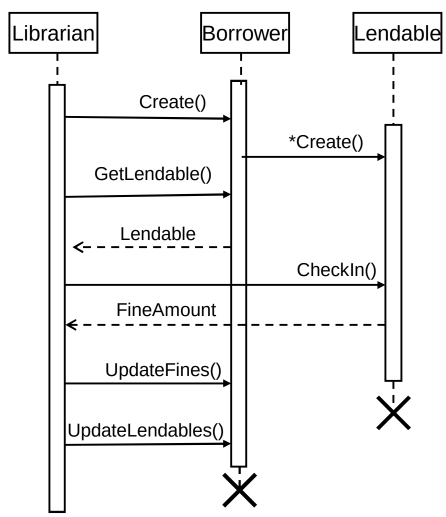
  Instead ask Book to check itself in and return amount of Fine if any.
- *Book*: If overdue calculate Fine, change status to IN and clear borrower and due date attributes and return Fine to Librarian.
- *Librarian*: Tell Borrower to update Fine Amount and update its set of Lendables.

BUT what if person returning book is not the person who borrowed it?

Check-In Scenario (2)

- At this stage Librarian class has become very large.
- Add a new Transaction superclass with Check Out and Check In as subclasses and have these take over many of Librarian’s responsibilities.
- Since Fines will need to be retrieved for billing it becomes clear that a Fines class is desirable and that the DB class should have responsibility for storing and retrieving these.

Check In Sequence Diagram



Updated Cards – Persistent classes

User Interacter

get employee ID	GUI subsystem
get user action	GUI subsystem
get lendable ID	GUI subsystem
display message	GUI subsystem
disp. OK/Cancel	GUI subsystem

DB

create	DBMS
get info DBMS	DBMS
get lendable	Lendable
get Borrower	Borrower, Lendable
put Lendable	Lendable
put Borrower	Borrower
put Fine	DBMS

Librarian

wait for user	
get/verify emp. ID	User Interacter, ID Verifier
know curr. borrower	DB
create/exec. Trans.	User Interacter, Transaction
get lendable ID	User Interacter
display error mess.	User Interacter

ID Verifier

verify	DB

Updated Cards – Transactions

Transaction
subclasses: Check In, Check Out
create
  know current borrower
execute

Check Out

create	
know current borrower	
know current lendable	DB
execute	
check borrower status	Borrower
check out lendable	Lendable
update borrower	Borrower
display errors	User Interacter
destroy	
put borrower	DB
put Lendable	DB

Check In

create	
know current borrower	
know current lendable	Borrower
get alternate borrower	DB
execute	
display errors	User Interacter
check in lendable	Lendable
create and store fine	Fine, DB
update borrower	Borrower
destroy	
put borrower	DB
put lendable	DB

Updated Cards – Lendable

Lendable		**Book, Video, Journal**	
subclasses: Book, Video, Journal		**superclass**: Lendable	
calculate due date	Date	calculate due date	Date
calculate fine	Date	calculate fine	
check out	Date		
check in			
know if overdue	Date		
know due date			
know borrower			
know in/out status			

Updated Cards – Borower, etc.

Borrower		**Fine**
create	List	know borrower
can borrow		know lendable
know total fine amount		know fine amount
know number of lendables	List	know due date
retrieve lendable from set	List	know returned date
check overdue lendable	Lendable	
add lendable to set	List	**Date**
delete lendable from set	List	know today's date
		subtract dates
		add days to date