CptS 487
Software Design and Architecture

Lesson 8

Object Modeling

WASHINGTON STATE UNIVERSITY
World Class. Face to Face.

Instructor: Bolong Zeng

Outline

- Use Case and Requirements Analysis
- Object Modeling
  - Three types of classes/objects:
    - Entity class/object
    - Boundary class/object
    - Control class/object
- And how to make use of these three types of classes.

Requirements and Use Case

- It's another topic entirely.
  - However, necessary to start the design process.

- Software requirements
  - Non-functional requirements
    - Mostly constraints and qualities
  - Functional requirements
    - Often modeled by Use Case

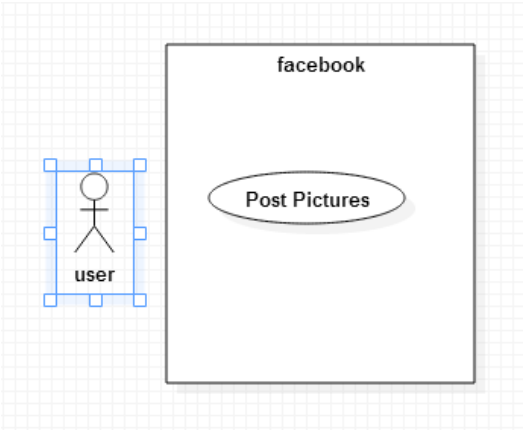
Requirements and Use Case

- Quote from Wikipedia:
  - In systems engineering and requirements engineering, a **non-functional requirement** is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions.
    - https://en.wikipedia.org/wiki/Non-functional_requirement

- Both are important. For now, we discuss Functional Requirements.
  - Defines what your software/system **IS**

Use Case

- Use Case
  - Looks at functional requirements from an external point of view. Describe a “feature”, for instance.
  - Involves two elements: an Actor, and a Use Case
  - Example:
    - Feature of facebook: “The user can post pictures to his own timeline”
    - Actor: User; Use Case: “Post Pictures”
    - UML Use Case Diagram:



- Actor
  - External user/party/device/system that interacts with your system.
- Use case
  - Internal function that the Actor performs with your system

Use Case

- A complete use case will also include a full template
  - See https://en.wikipedia.org/wiki/Use_case
  - The Example section

From Analysis to Design

- Typical process:
  - Identify the requirements in Use Case form.
  - Perform Object Modeling.

- For now, simply think of the “functional requirements” as features
  - Find possible actors, and determine what they are able to/supposed to do with your system.
  - Hint: In a game, other than player, what might be a good candidate for Actor?

Activities during Object Modeling

Main goal: Find the important abstractions

- Steps during object modeling
  1. Class identification
  2. Find the attributes
  3. Find the operations
  4. Find the associations between classes
- Order of steps
  - Goal: get the desired abstractions
  - Order of steps is secondary
- What happens if we find the wrong abstractions?
  - We iterate and revise the model.

Class Identification

Class identification is crucial to object-oriented modeling

- Helps to identify the important entities of a system

- Approaches
  - Application domain approach
    - Ask application domain experts to identify relevant abstractions
  - Syntactic approach
    - Start with use cases
    - Analyze the text to identify the objects
    - Extract participating objects from flow of events
  - Design patterns approach
    - Identify relevant abstractions that can be reused (apply design knowledge)

Class identification is a Hard Problem

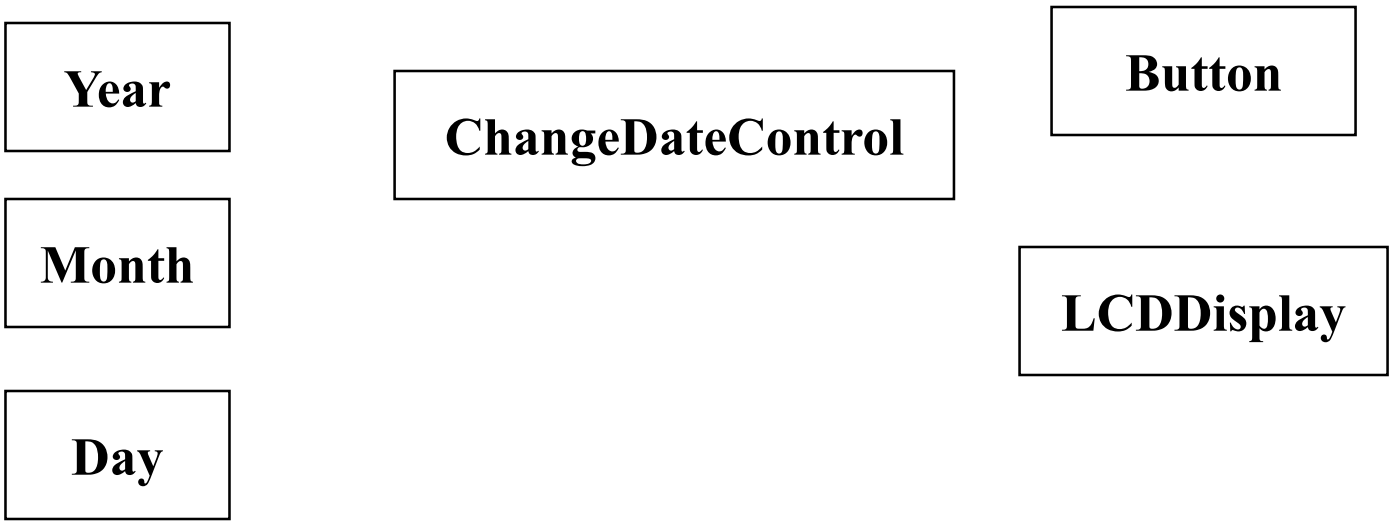
- One problem: Definition of the system boundary:
  - Which abstractions are outside, which abstractions are inside the system boundary?
    - Actors are outside the system
    - Classes/Objects are inside the system
- An other problem: Classes/Objects are not just found by taking a picture of a scene or domain
  - The application domain has to be analyzed
  - Depending on the purpose of the system different objects might be found
    - Scenarios and use cases => Functional model.

There are different types of Objects

- Entity Objects
  - Represent the persistent information tracked by the system (Application domain objects, also called “Business objects”)
- Boundary Objects
  - Represent the interaction between the user and the system
- Control Objects
  - Represent the control tasks performed by the system.

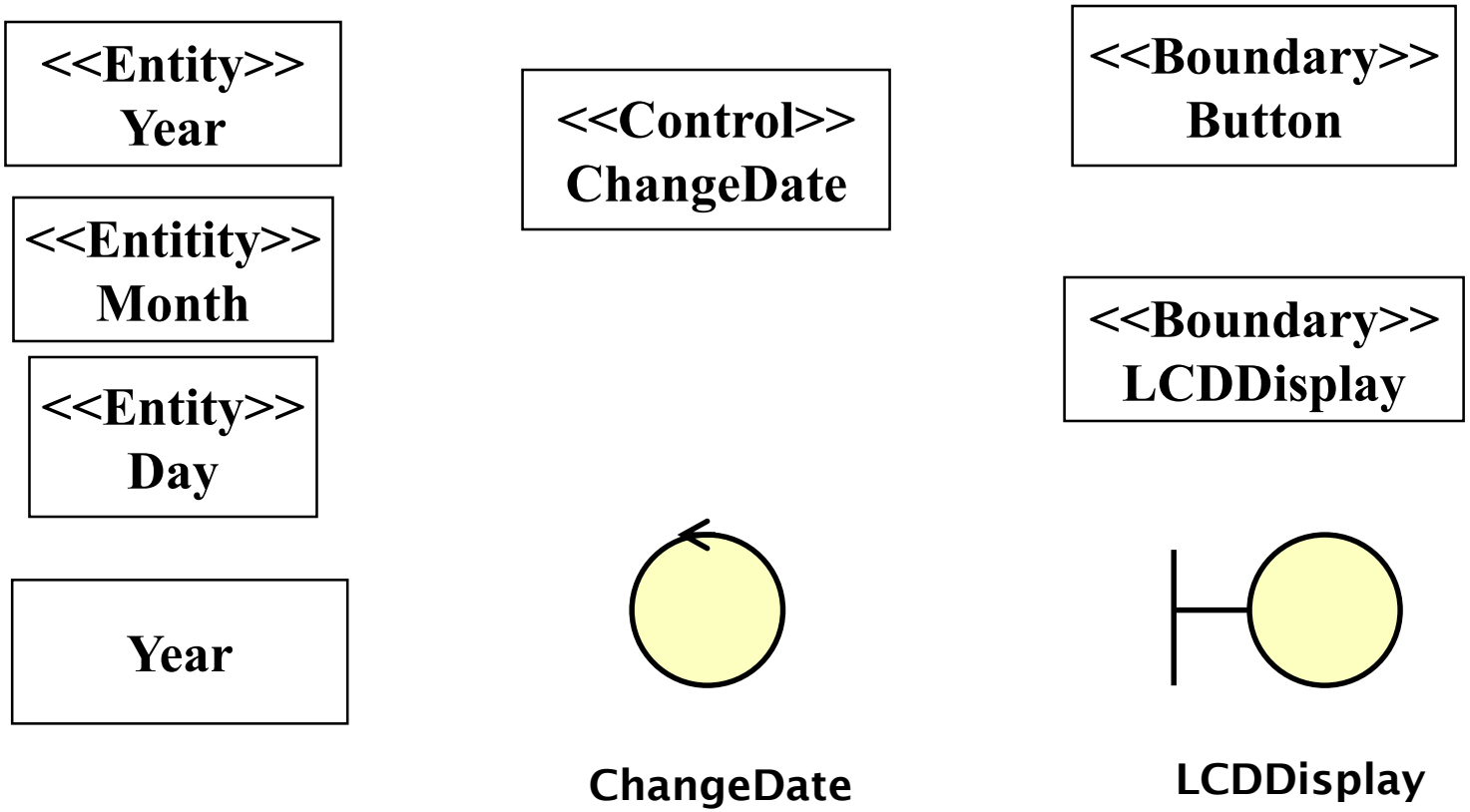
Example: 2BWatch Modeling

To distinguish different object types in a model we use the UML Stereotype mechanism



Entity Objects **Control Object** **Boundary Objects**

Naming Object Types in UML



Finding Participating Objects in Use Cases

- Pick a use case and look at flow of events
- Do a textual analysis (noun-verb analysis)
  - Nouns are candidates for objects/classes
  - Verbs are candidates for operations
  - This is also called Abbott's Technique

- After objects/classes are found, identify their types
  - Identify real world entities that the system needs to keep track of (FieldOfficer Entity Object)
  - Identify real world procedures that the system needs to keep track of (EmergencyPlan Control Object)
  - Identify interface artifacts (PoliceStation Boundary Object).

Identifying Entity Objects

- Syntactical investigation with Abbot‘s technique:
  - Flow of events in use cases
  - Problem statement

Example for using the Technique

Example use case:

Use Case Name Borrow Books

Participating Actors: borrower, librarian

Entry condition:

- The borrower has the ID card of this library

Exit condition:

- The borrower left with the books he wanted.

Example for using the Technique

Example use case: Borrow Books

Actor steps	System steps
1. **A borrower brings a few books to the counter.**	
2. **The librarian scans the borrower's ID**	
4. The librarian scans the books' bar code	
	3. **The system checks the fine status whether excess over $20.**
	5. **The system checks the borrowing limit of the books whether has been reached.**
	6. **The system approves the loan for the borrower.**
	7. **The system prints out the statement of due dates for the books.**

Alternative Courses:
Line 3: The librarian tells the borrower to pay the fine. Use case canceled.
Line 5: The system notifies the librarian that the limit borrowing amount of book is over. Use case cancelled.

Mapping grammatical constructs to model components (Abbot's Technique)

Example	*Grammatical construct* →	*UML model component*
Student, book		improper noun
Scan, approve		verb
Is a		being verb
has an		having verb
must be		modal verb
borrowing limit		noun phrase
book's barcode		possessive noun

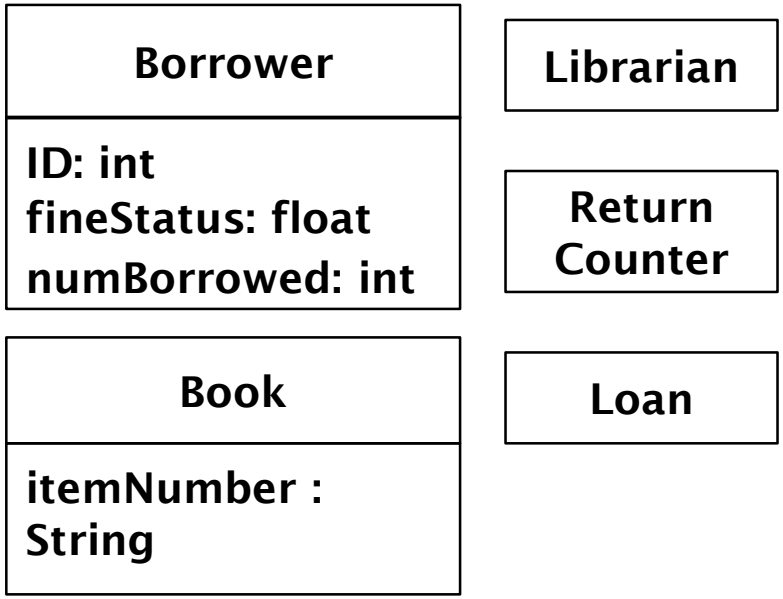
Mapping grammatical constructs to model components (Abbot's Technique)

Example	*Grammatical construct*	→ *UML model component*
Student, book	improper noun	class
Scan, approve	verb	operation, association
Is a	being verb	inheritance
has an	having verb	aggregation
must be	modal verb	constraint
borrowing limit	noun phrase	attribute
book's barcode	possessive noun	attribute

Generating a Class Diagram from Flow of Events

Actor steps

1. A borrower brings a few books to the counter.
2. The librarian scans the borrower's ID
4. The librarian scans the books' bar code



System steps

3. The system checks the fine status whether excess over $20.
5. The system checks whether the borrowing limit has been reached for the borrower.
6. The system approves the loan for the borrower.
7. The system displays and prints out due dates for the books.

Heuristics to Identify Attributes and Associations

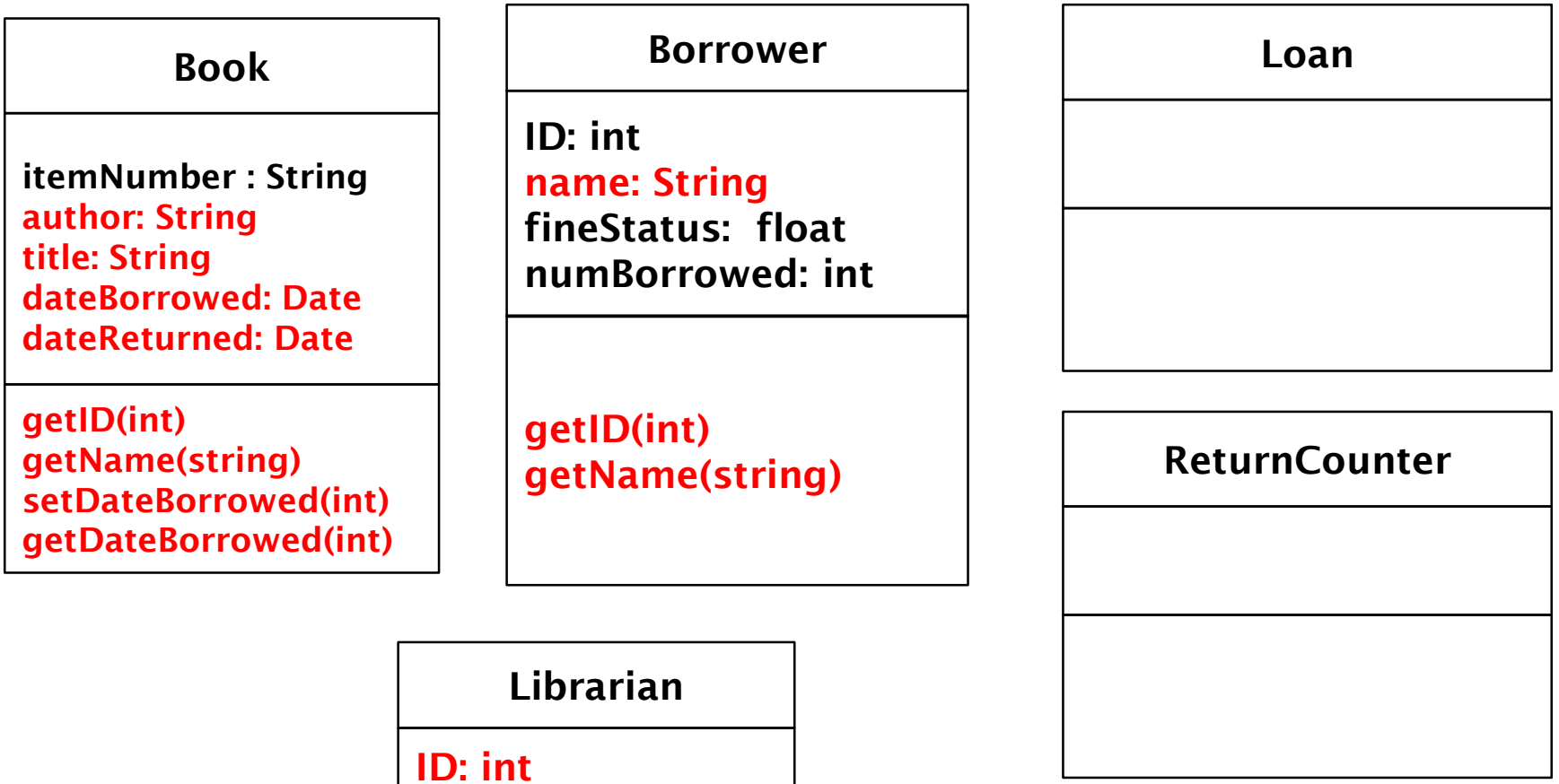
1. An attribute represents a single property of an object.
   - Attributes have simple types and represent atomic concepts.
2. Associations represent the relationships between complex concepts (i.e., objects)
   - Complex concepts are represented as separate objects
   - Nouns referring to collections are associations

Identifying Entity Objects (revisited)

- Syntactical investigation with Abbot's technique:
  - Flow of events in use cases
  - Problem statement

- Use other knowledge sources:
  - Application knowledge: End users and experts know the abstractions of the application domain
  - Solution knowledge: Abstractions in the solution domain
  - General world knowledge: Your generic knowledge and intuition

Borrow Books Example - Classes



Identifying Boundary Objects

- Boundary objects represent the system interface with the actors
  - Boundary classes handle the communication between actors and the control and entity objects
  - Each actor interacts with at least one boundary object in each use case.

Heuristics for Identifying Boundary Objects

- Identify user interface controls that the user needs to initiate the usecase.
- Identify forms the users needs to enter data into the system (e.g., LoanEntryWindow).
- Identify notices and messages the system uses to respond to the user.
- When multiple actors are involved in a use case, identify actor terminals.
- Do not model the visual aspects of the interface with boundary objects (user mock-ups are better
- suited for that).
- Always use the end user's terms for describing interfaces; do not use terms from the solution or implementation domains.

Borrow Books Example - Classes

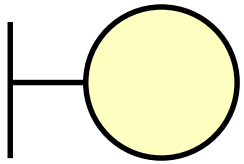
Book
itemNumber : String author: String title: String dateBorrowed: Date dateReturned: Date
getID(int) getName(string) setDateBorrowed(int) getDateBorrowed(int)

Borrower
ID: int name: String fineStatus: float numBorrowed: int
getID(int) getName(string) setNumBorrowed(int) getnumBorrowed(int)

Loan

ReturnCounter

Librarian
ID: int name: String
getID(int) getName(string)



LoanEntryWindow

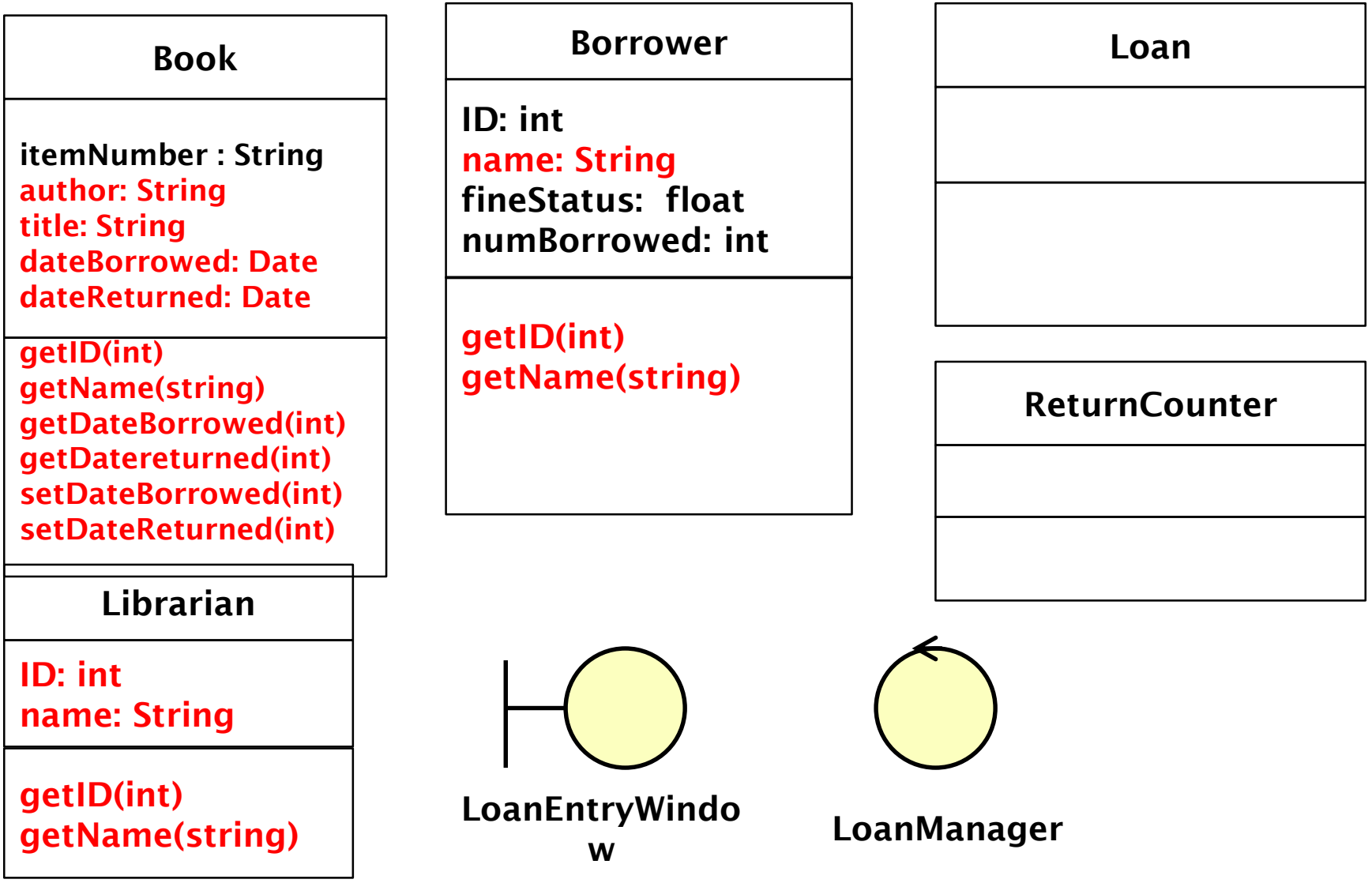
Identifying Control Objects

- Control objects are responsible for coordinating boundary and entity objects.
  - Collect information from the boundary objects and dispatch it to the entity objects
- Control objects do not usually correspond to an instance in the real world.
- Control objects are created at the beginning of a use case and ceases at the end.

Heuristics for Identifying Control Objects

- Identify one control object per use case (e.g., LoanManager).
- The life span of a control object should cover the extent of the use case or the extent of a user session

Borrow Books Example - Classes



Modeling Interactions Among Objects

- A sequence diagram:
  - shows how the behavior of the use case is distributed among objects
  - sequence of interactions among objects
  - lifetime of objects

- Sequence diagrams are used to help to identify new participating objects and missing behavior.

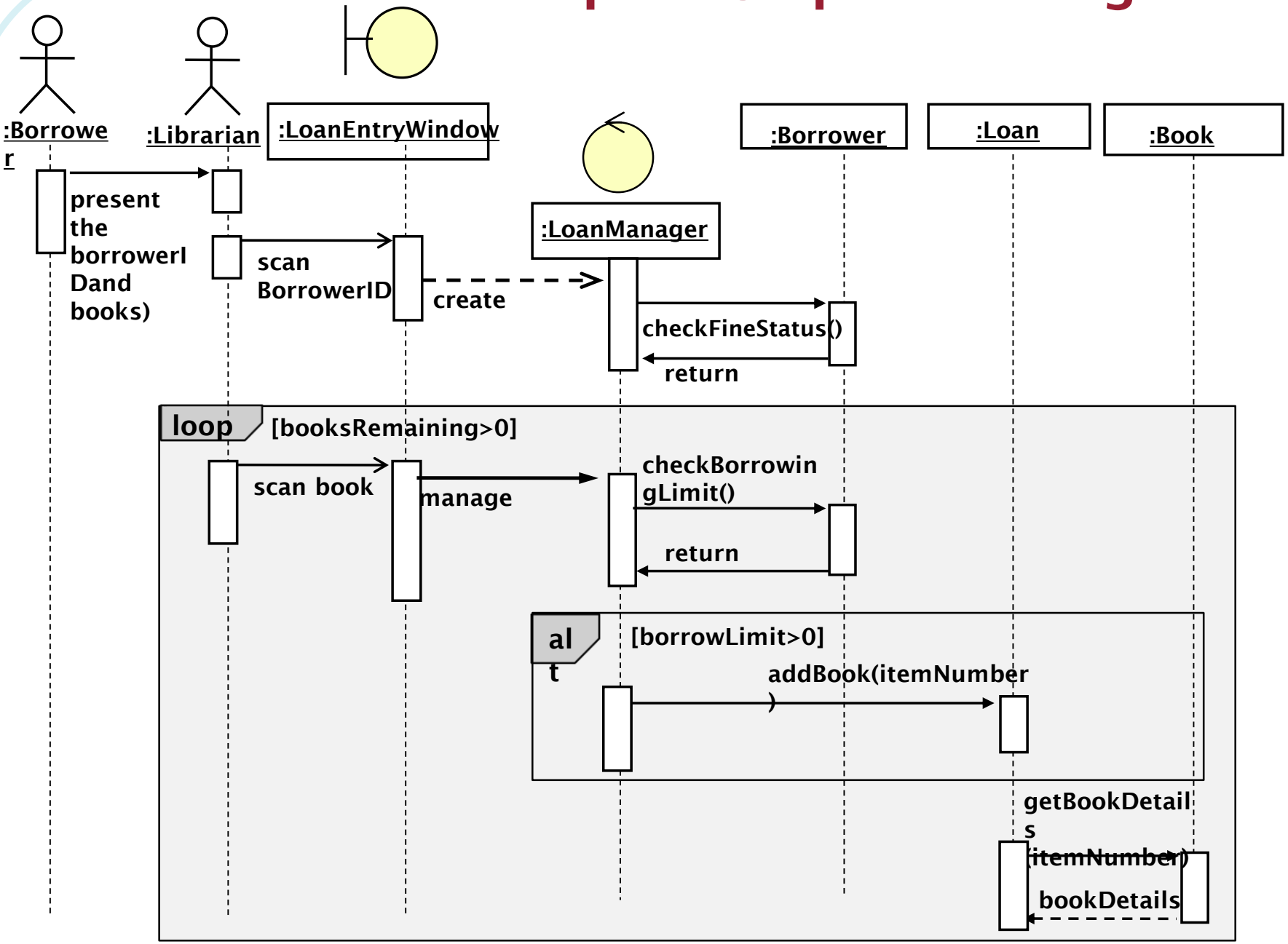
Heuristics for Sequence Diagrams

- Layout:
  1st column: Should be the actor of the use case
  2nd column: Should be a boundary object
  3rd column: Should be the control object that manages the rest of the use case

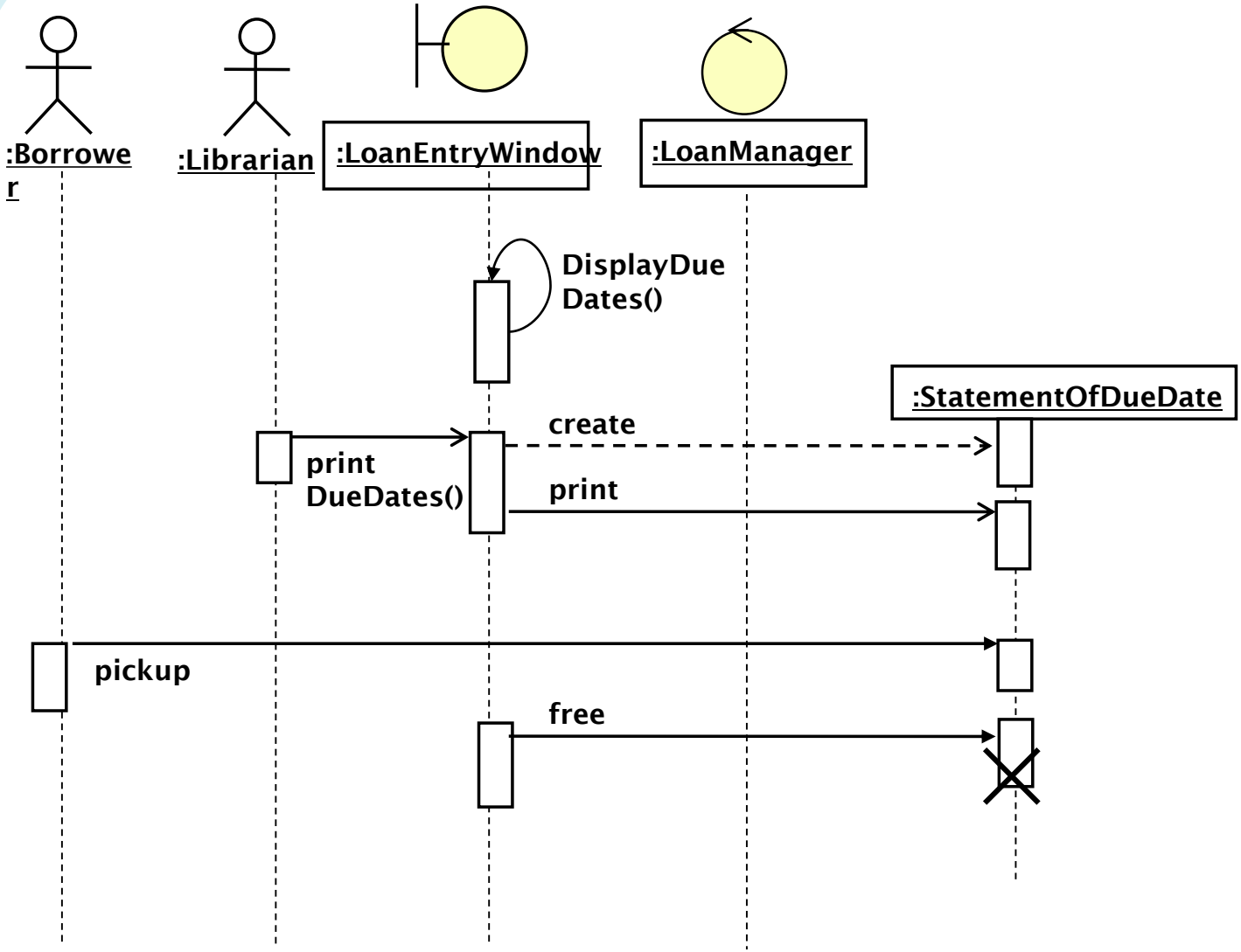
- Creation of objects:
  — Create control objects at beginning of event flow

- Access of objects:
  — Entity objects can be accessed by control and boundary objects
  — Entity objects should not access boundary or control objects.

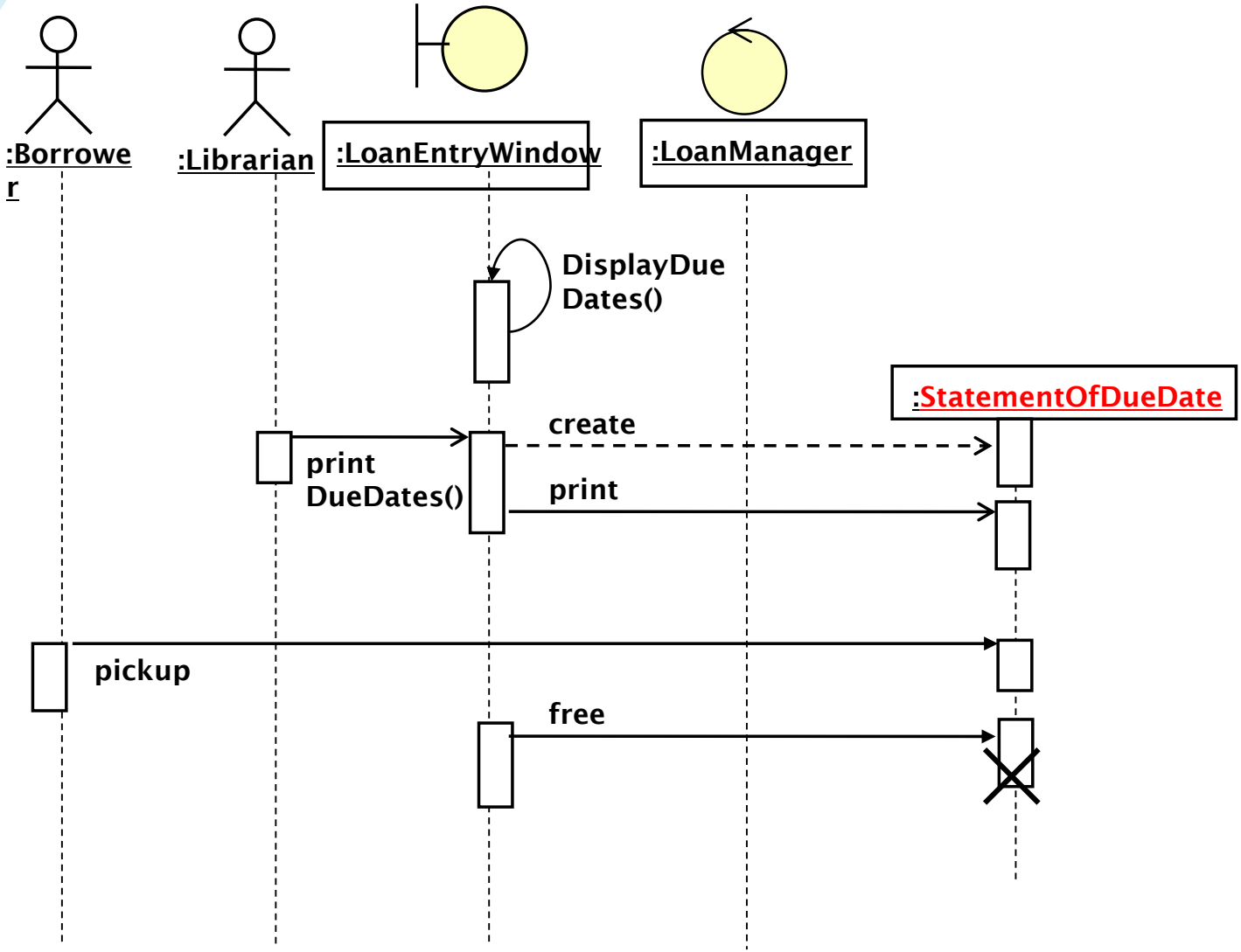
Borrow Books Example – Sequence Diagram



Borrow Books Example – Sequence Diagram



Borrow Books Example – Sequence Diagram



Borrow Books Example - Classes

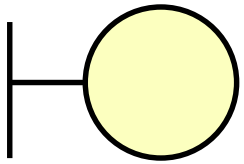
Book
itemNumber : String author: String title: String dateBorrowed: Date dateReturned: Date
getID(int) getName(string) getDateBorrowed(int) getDatereturned(int) setDateBorrowed(int) setDateReturned(int)

Borrower
ID: int name: String fineStatus: float numBorrowed: int
getID(int) getName(string) setNumBorrowed(int) getnumBorrowed(int)

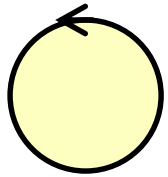
Loan
numberofBooks:int
setNumberofBooks(int) getNumberofBooks(int)

ReturnCounter

Librarian
ID: int name: String
getID(int) getName(string)



LoanEntryWindow



LoanManager

StatementOfDueDate