Introduction to UML

CSE 333: Software Engineering

About this lecture…

- Will attempt to introduce you to UML
- Not possible to teach everything
- Requires that you study on you own after
- Goal is to get you familiar
- Make use of on-line tutorials, books, etc.

Introduction to UML

- What is UML?
- Motivations for UML
- Types of UML diagrams
- UML syntax
- Descriptions of the various diagram types
  - Rational Rose and UML
- UML pitfalls

What is UML?

- A standardized, graphical “modeling language” for communicating software design.
- Allows implementation-independent specification of:
  - user/system interactions (required behaviors)
  - partitioning of responsibility (OO)
  - integration with larger or existing systems
  - data flow and dependency
  - operation orderings (algorithms)
  - concurrent operations
- UML is not “process”. (That is, it doesn’t tell you how to do things, only what you should do.)

Motivations for UML

- UML is a fusion of ideas from several precursor modeling languages.
- We need a modeling language to:
  - help develop efficient, effective and correct designs, particularly Object Oriented designs.
  - communicate clearly with project stakeholders (concerned parties: developers, customer, etc.).
  - give us the “big picture” view of the project.

Types of UML diagrams

- There are different types of UML diagram, each with slightly different syntax rules:
  - use cases.
  - class diagrams.
  - sequence diagrams.
  - package diagrams.
  - state diagrams
  - activity diagrams
  - deployment diagrams.

UML syntax, 1

- Actors: a UML actor indicates an interface (point of interaction) with the system.
  - We use actors to group and name sets of system interactions.
  - Actors may be people, or other systems.
  - An actor is NOT part of the system you are modeling. An actor is something external that your system has to deal with.
- Boxes: boxes are used variously throughout UML to indicate discrete elements, groupings and containment.

UML syntax, 2

- Arrows: arrows indicate all manner of things, depending on which particular type of UML diagram they’re in. Usually, arrows indicate flow, dependency, association or generalization.
- Cardinality: applied to arrows, cardinalities show relative numerical relationships between elements in a model: 1 to 1, 1 to many, etc.

UML syntax, 3

- Constraints: allow notation of arbitrary constraints on model elements. Used, for example, to constrain the value of a class attribute (a piece of data).
- Stereotypes: allow us to extend the semantics of UML with English. A stereotype is usually a word or short phrase that describes what a diagram element does. That is, we mark an element with a word that will remind us of a common (stereotypical) role for that sort of thing. Stereotypes should always be applied consistently (with the same intended meaning in all instances).

UML pitfalls, 1

- UML is a language, with a (reasonably) rigorous syntax and accepted semantics; that is, the diagrams have a meaning. Thus you have to be careful that the meaning of your diagram is what you intended.
- However, the semantics of UML are less well-defined than a programming language (where the semantics are defined by the compiler). Thus there is some leeway to use UML your own way: but you must be consistent in what you mean by the things you draw.

UML pitfalls, 2

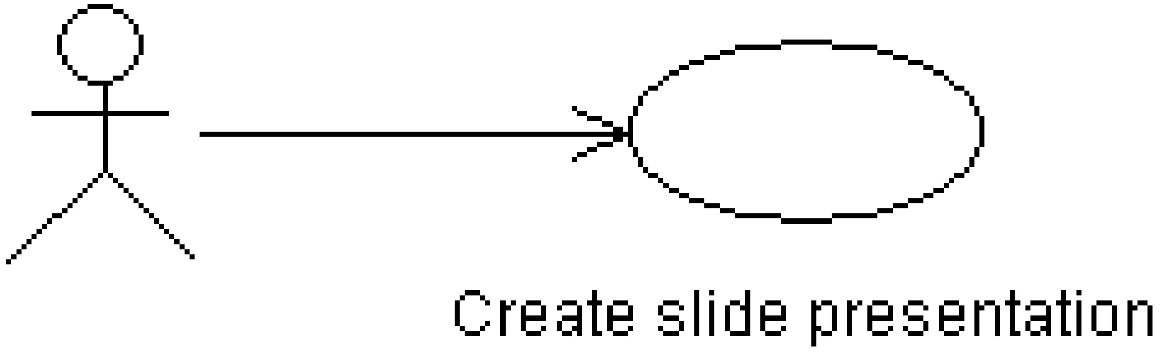
- Arrow happiness: people tend to draw arrows (associations) everywhere in their diagrams, inconsistently without much regard for the UML meaning of a given arrow.
- Diagram fever: it's easy to do too many diagrams. The trick is to get the correct granularity. Eg, the requirements document should leave implementation detail to the architecture.
- General loopiness: be careful about slapping together UML diagrams, or doing a diagram without thoroughly understanding your system. You should always be able to give a clear and concise explanation of your diagram, and why you did it that way.

UML diagrams: use cases

- A use case encodes a typical user interaction with the system. In particular, it:
  - captures some user-visible function.
  - achieves some concrete goal for the user.
- A complete set of use cases largely defines the requirements for your system: everything the user can see, and would like to do.
- The granularity of your use cases determines the number of them (for you system). A clear design depends on showing the right level of detail.
- A use case maps actors to functions. The actors need not be people.

Use case examples, 1

(High-level use case for powerpoint.)

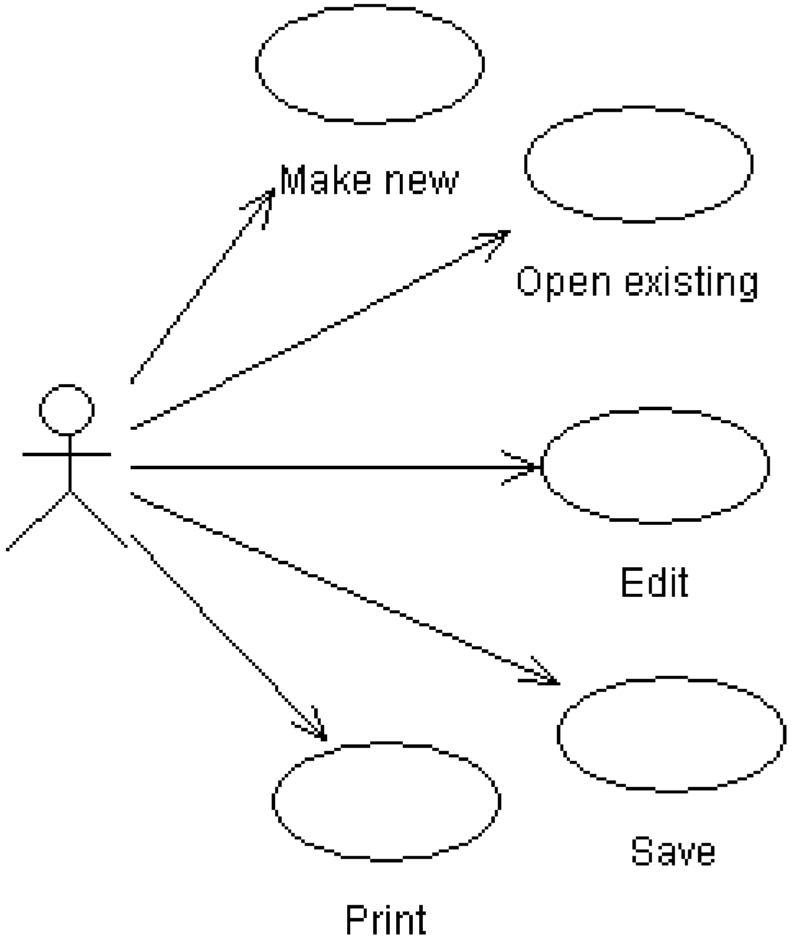


About the last example...

- Although this is a valid use case for powerpoint, and it completely captures user interaction with powerpoint, it’s too vague to be useful.

Use case examples, 2

(Finer-grained use cases for powerpoint.)

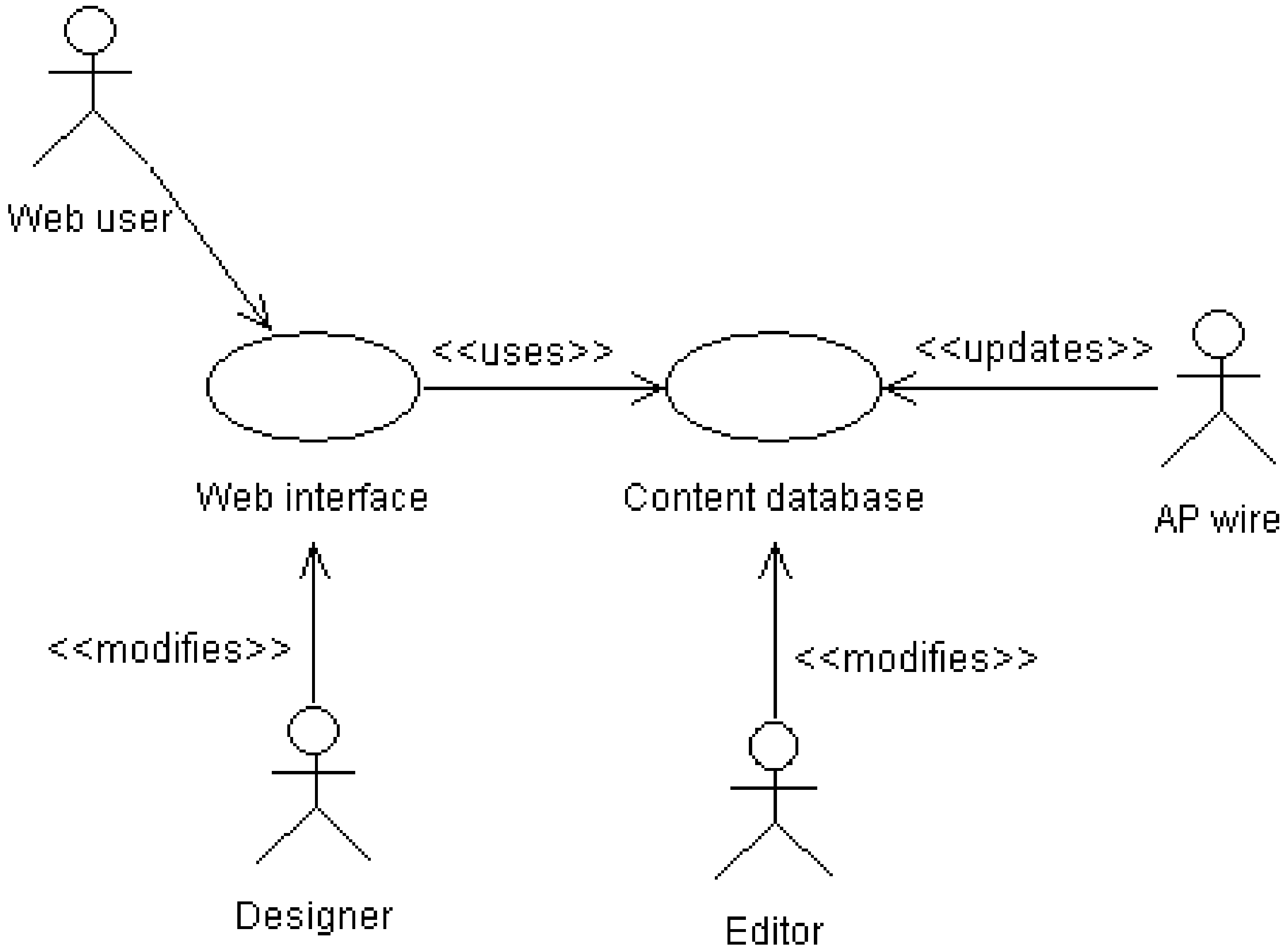


About the last example...

- The last example gives a more useful view of powerpoint (or any similar application).
- The cases are vague, but they focus your attention the key features, and would help in developing a more detailed requirements specification.
- It still doesn’t give enough information to characterize powerpoint, which could be specified with tens or hundreds of use cases (though doing so might not be very useful either).

Use case examples, 3

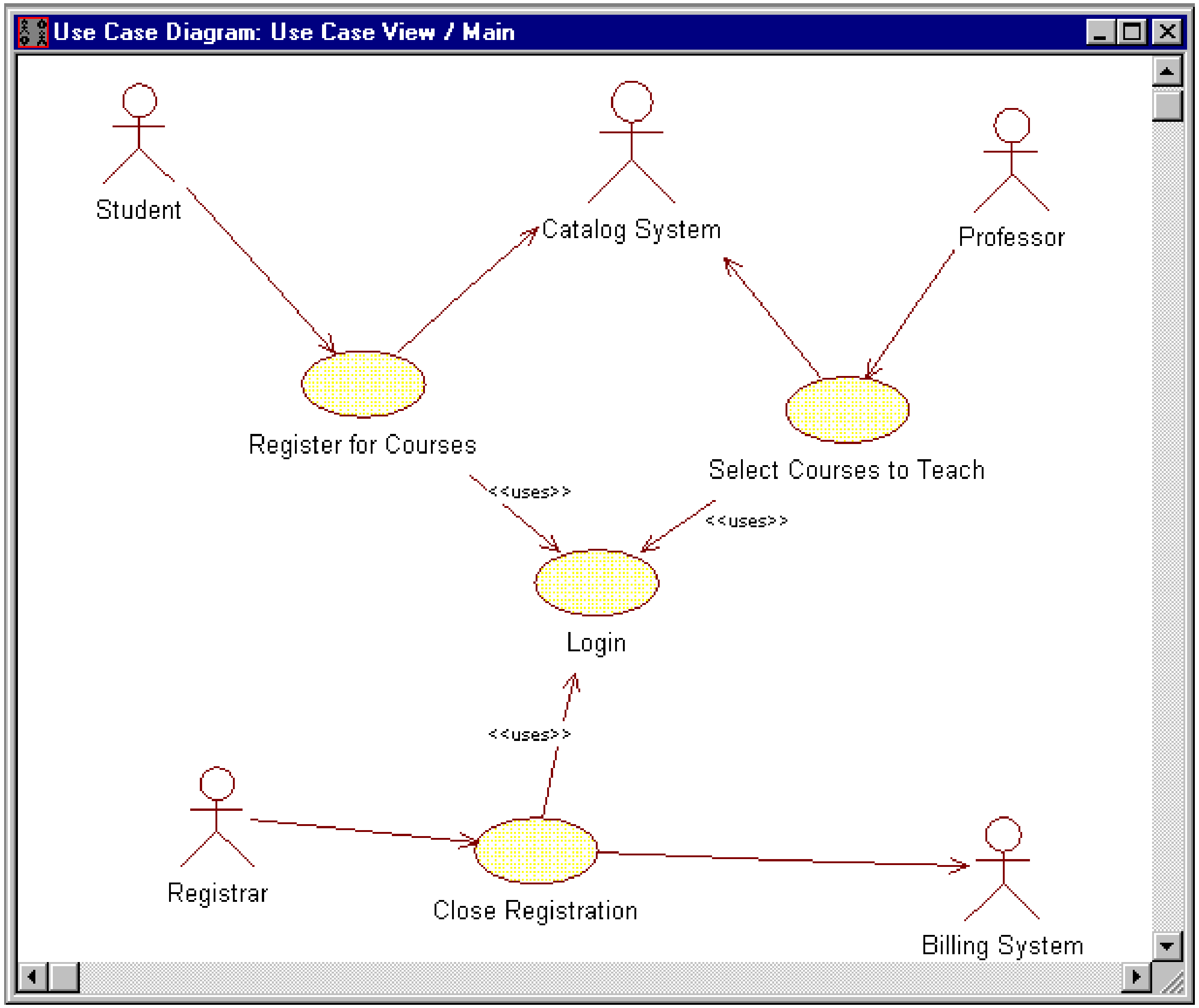
(Relationships in a news web site.)



About the last example...

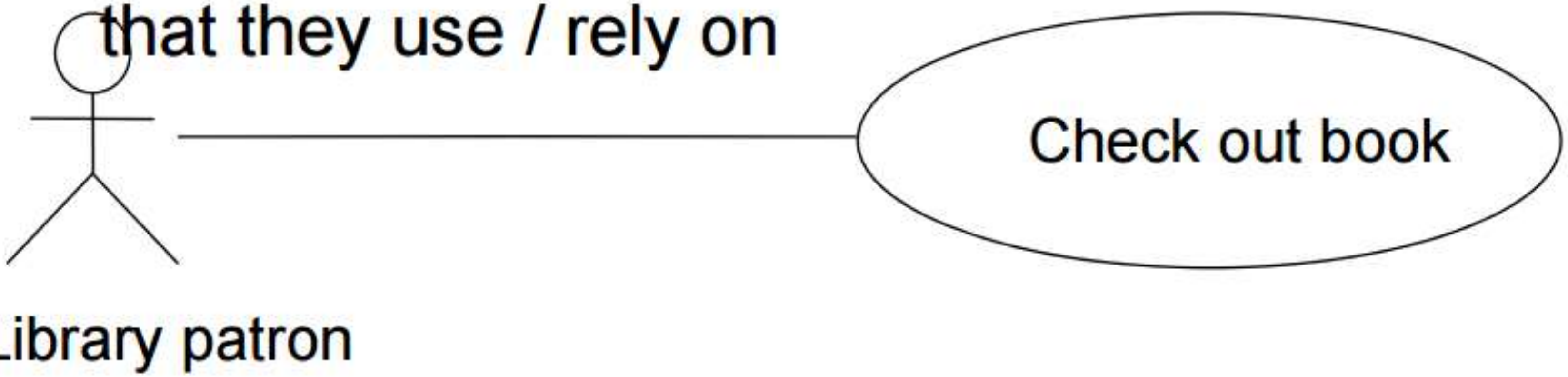
- The last is more complicated and realistic use case diagram. It captures several key use cases for the system.
- Note the multiple actors. In particular, ‘AP wire’ is an actor, with an important interaction with the system, but is not a person (or even a computer system, necessarily).
- The notes between << >> marks are *stereotypes:* identifiers added to make the diagram more informative. Here they differentiate between different roles (ie, different meanings of an arrow in this diagram).

Use Case Relationships



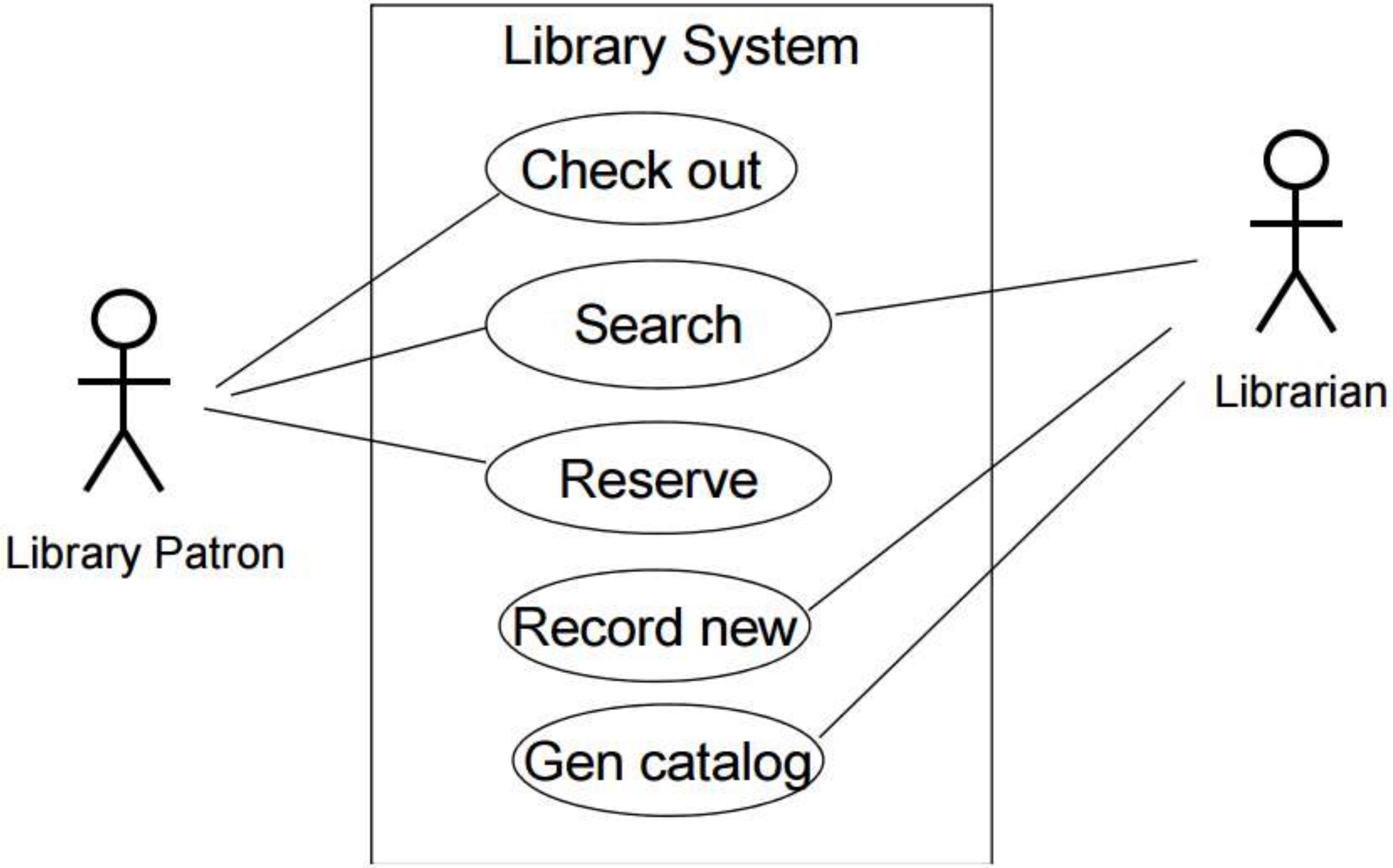
The overall list of your system's use cases can be drawn as high-level diagrams, with:

- actors as stick-men, with their names (nouns)
- use cases as ellipses with their names (verbs)
- line associations, connecting an actor to a use case in which that actor participates
- use cases can be connected to other cases that they use / rely on



An example

In a library system Library patron can search a book and reserve it for him/her. Then he/she needs to visit Library desk and check out by himself. If he/she does not find the book then he/she can request librarian to check the book in the system. Librarian also can insert the new record of the book and generate a catalog.


Library System
Check out
Search
Reserve
Record new
Gen catalog
Library Patron
Librarian

UML diagrams: class diagram

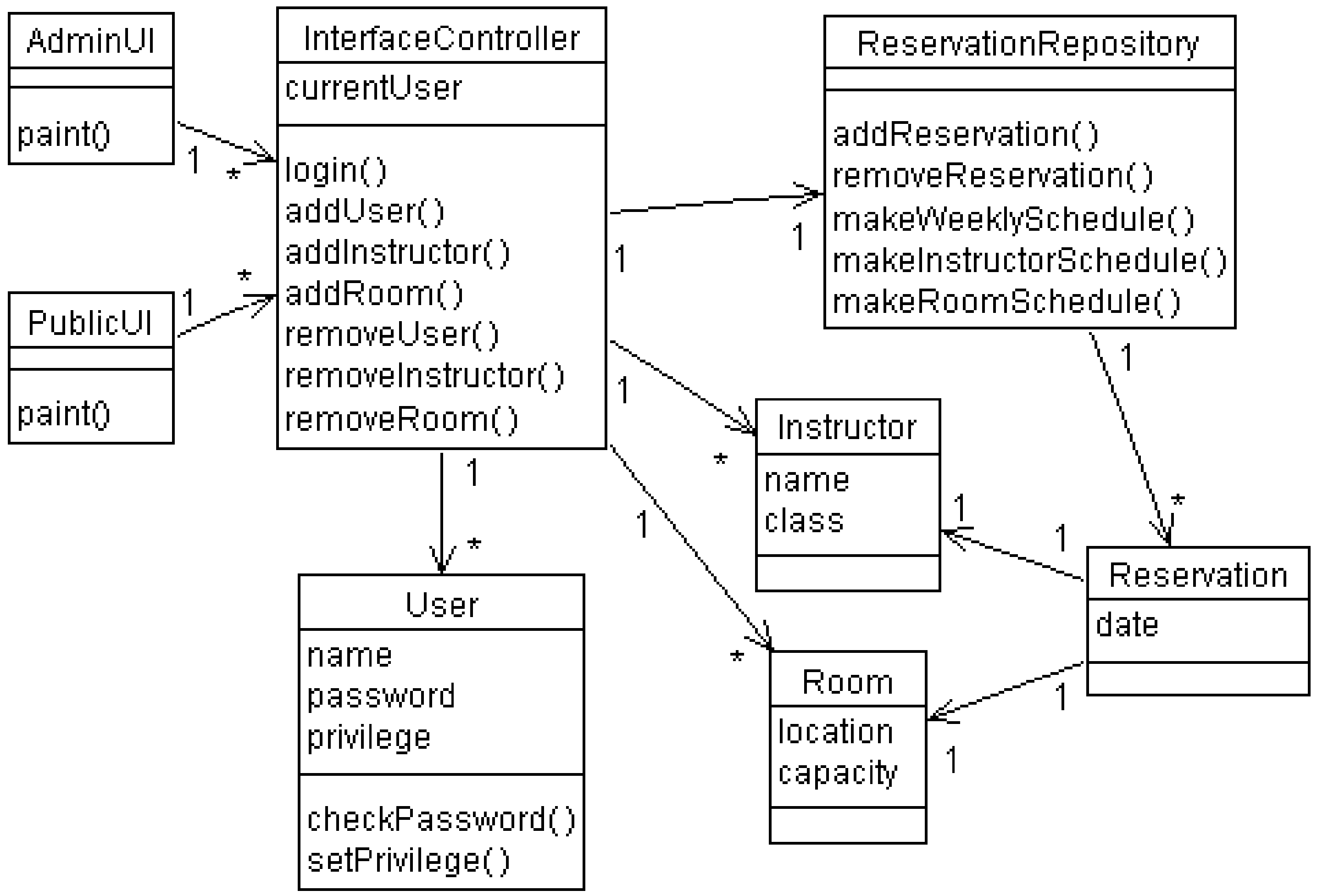
- Motivated by Object-Oriented design and programming (OOD, OOP).
- A class diagram partitions the system into areas of responsibility (classes), and shows “associations” (dependencies) between them.
- Attributes (data), operations (methods), constraints, part-of (navigability) and type-of (inheritance) relationships, access, and cardinality (1 to many) may all be noted.

Class diagram "perspective"

- Class diagrams can make sense at three distinct levels, or perspectives:
  - Conceptual: the diagram represents the concepts in the project domain. That is, it is a partitioning of the relevant roles and responsibilities in the domain.
  - Specification: shows interfaces between components in the software. Interfaces are independent of implementation.
  - Implementation: shows classes that correspond directly to computer code (often Java or C++ classes). Serves as a blueprint for an actual realization of the software in code.

Class diagram examples

(A classroom scheduling system: specification perspective.)

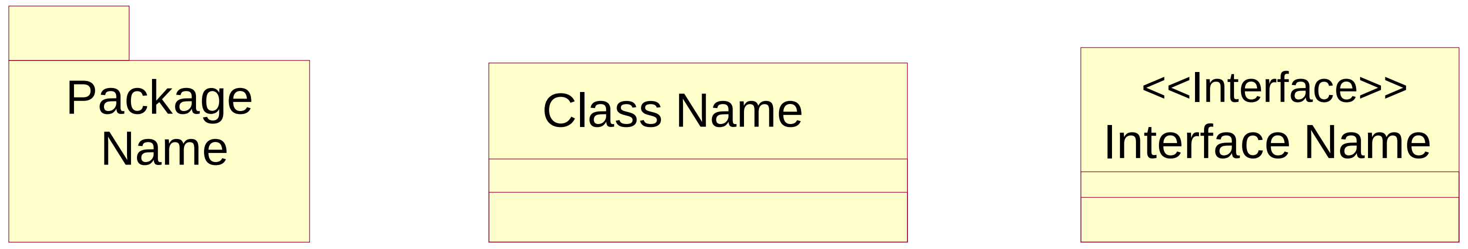


About the last example...

- Each box is a class, with necessary attributes and operations specified.
- Navigability arrows show which classes can reference which others.
- Cardinality marked in bi-directional manner on arrows.
- The classes together represent the complete system; thus the classes are a *partitioning* of the system.

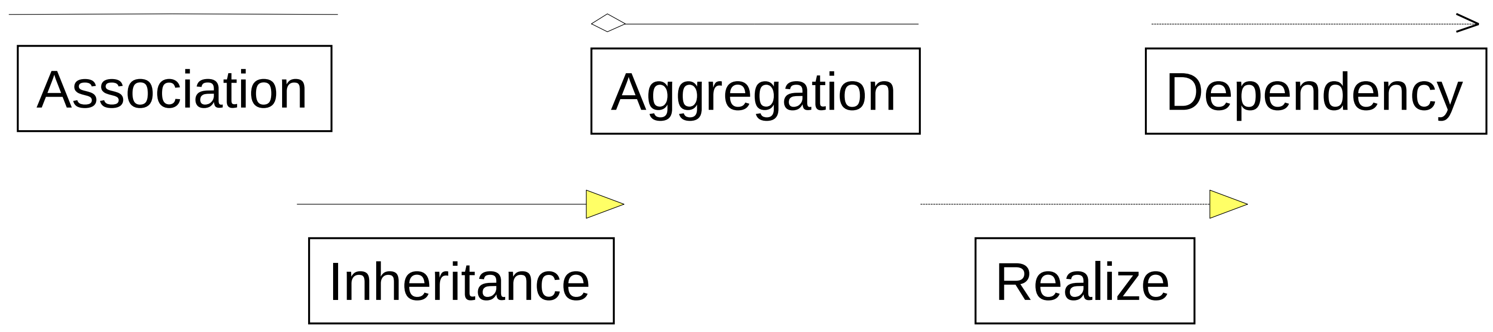
What is a Class Diagram?

- A class diagram is a view of the static structure of a system
  - Models contain many class diagrams
- Class diagrams contain:
  - Packages, classes, interfaces, and relationships
- Notation:



Relationships

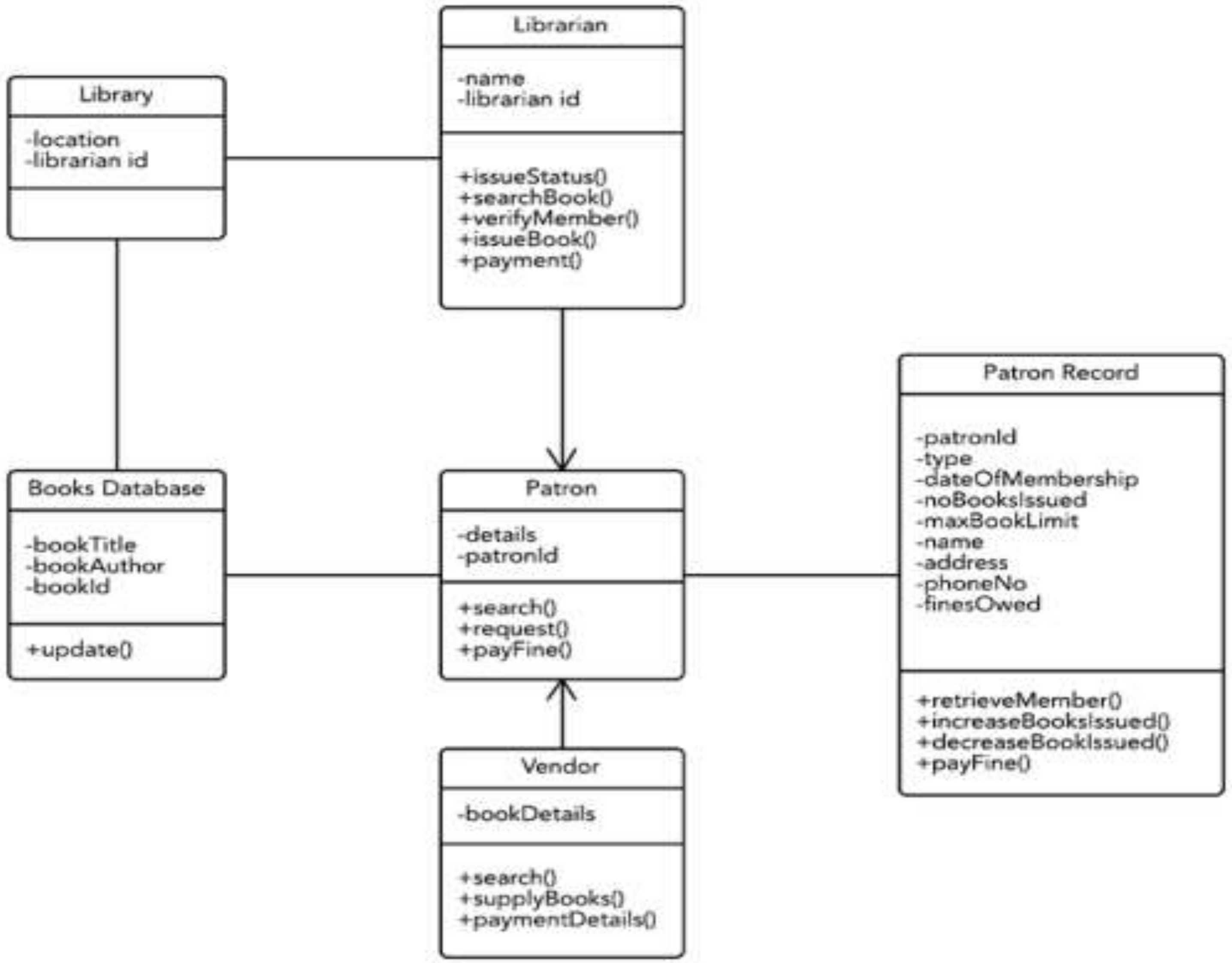
- Class diagrams may contain the following relationships:
  - Association, aggregation, dependency, realize, and inheritance
- Notation:



Multiplicity Indicators

- Each end of an association or aggregation contains a multiplicity indicator
  - Indicates the number of objects participating in the relationship

1	Exactly one
0..*	Zero or more
1..*	One or more
0..1	Zero or one
2..7	Specified range

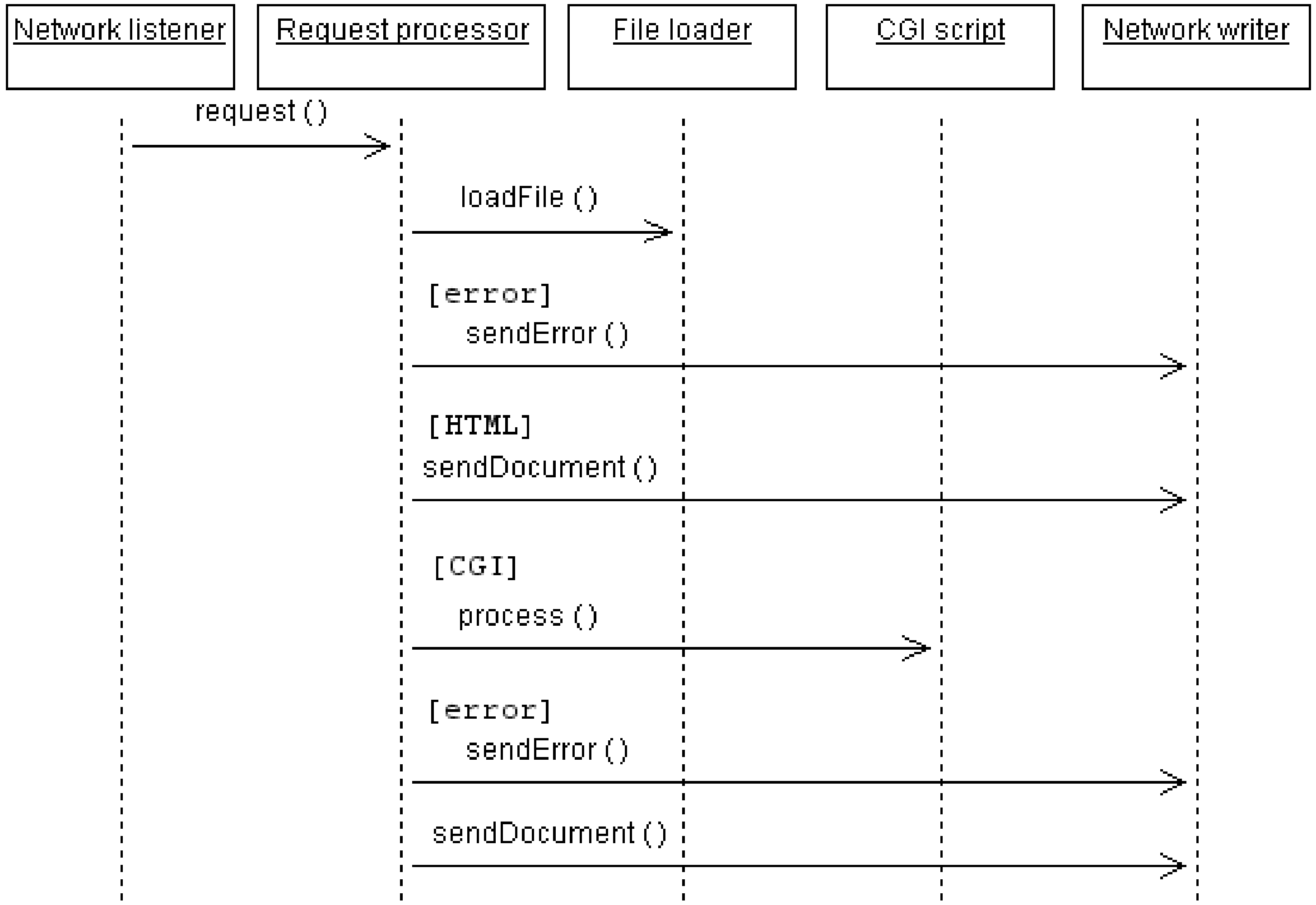

Library
-location
-librarian id
Librarian
-name
-librarian id
+issueStatus()
+searchBook()
+verifyMember()
+issueBook()
+payment()
Patron Record
-patronId
-type
-dateOfMembership
-noBooksIssued
-maxBookLimit
-name
-address
-phoneNo
-finesOwed
+retrieveMember()
+increaseBooksIssued()
+decreaseBookIssued()
+payFine()
Books Database
-bookTitle
-bookAuthor
-bookId
+update()
Patron
-details
-patronId
+search()
+request()
+payFine()
Vendor
-bookDetails
+search()
+supplyBooks()
+paymentDetails()

Sequence Diagrams

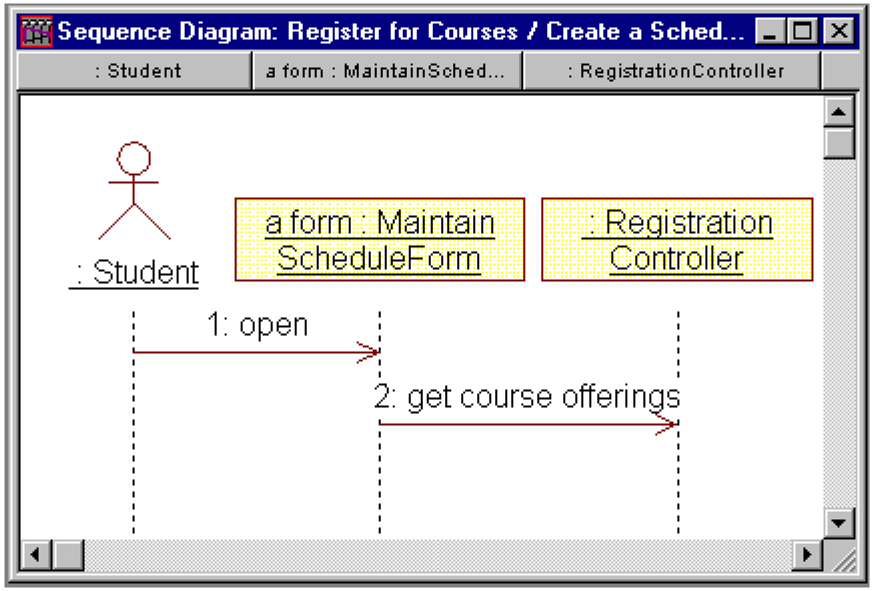
UML diagrams: sequence diagram

- Sequence diagram describe algorithms, though usually at a high level: the operations in a useful sequence diagram specify the “message passing” (method invocation) between objects (classes, roles) in the system.
- The notation is based on each object’s life span, with message passing marked in time-order between the objects. Iteration and conditional operations may be specified.
- May in principle be used at the same three levels as class diagrams, though the specification level will usually be most useful. (At the implementation level, you might better use pseudocode.)

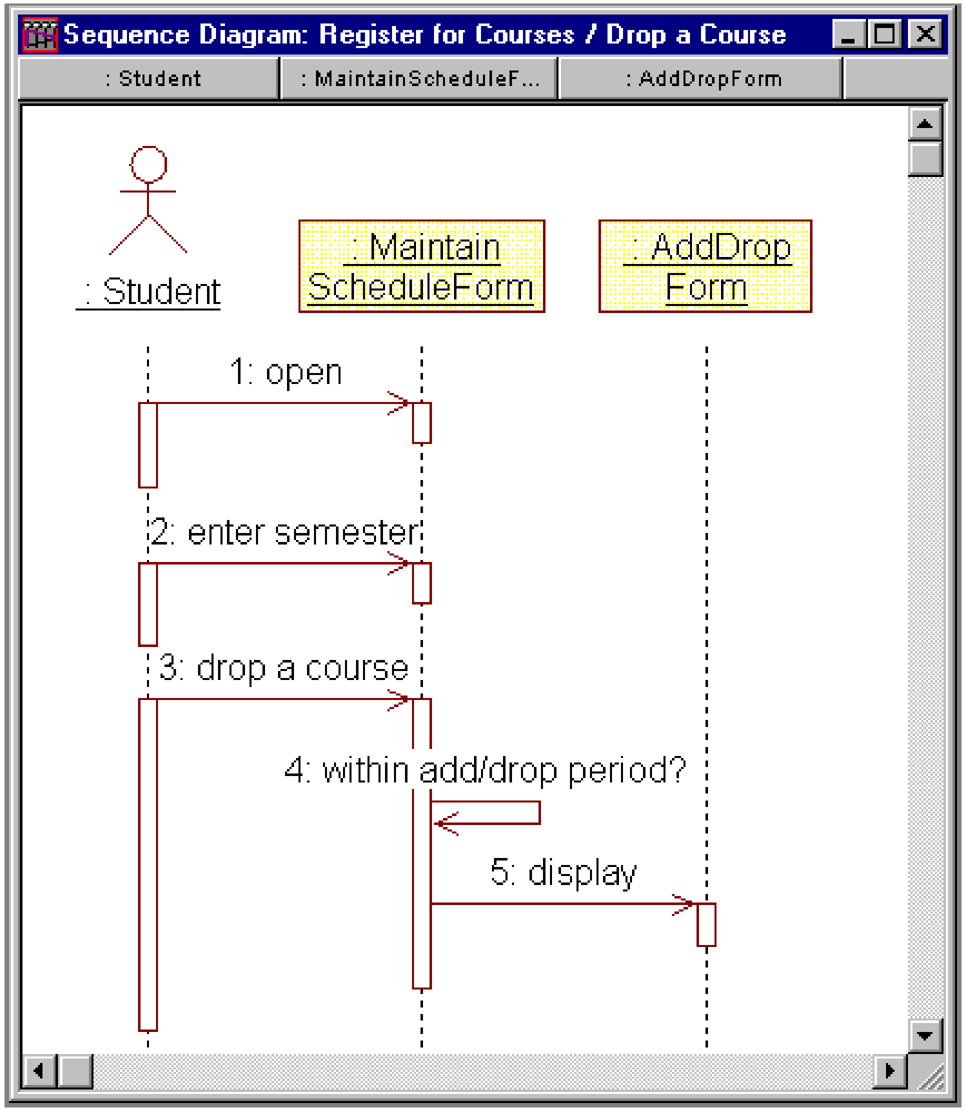
Sequence diagram example



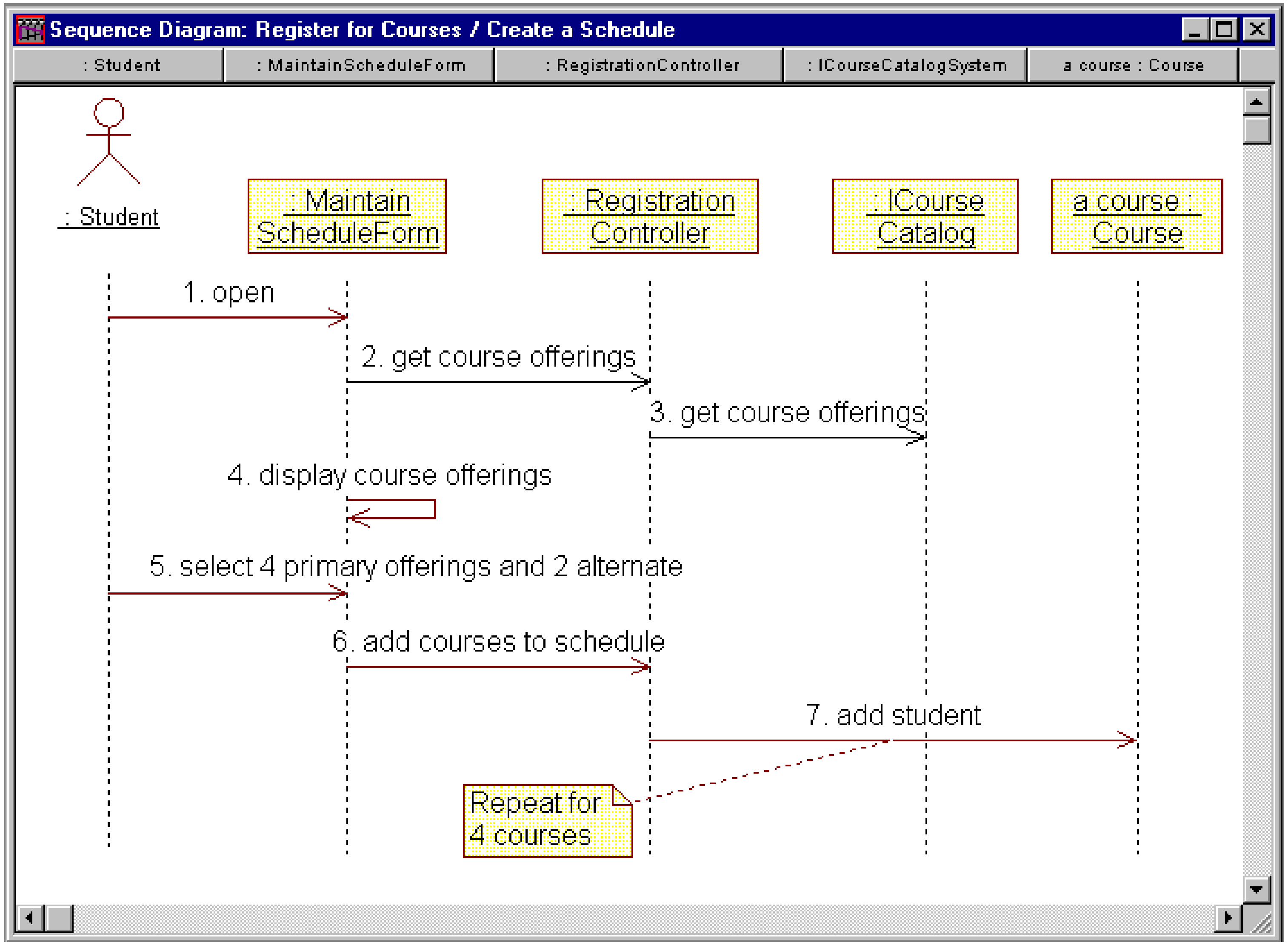
Messages

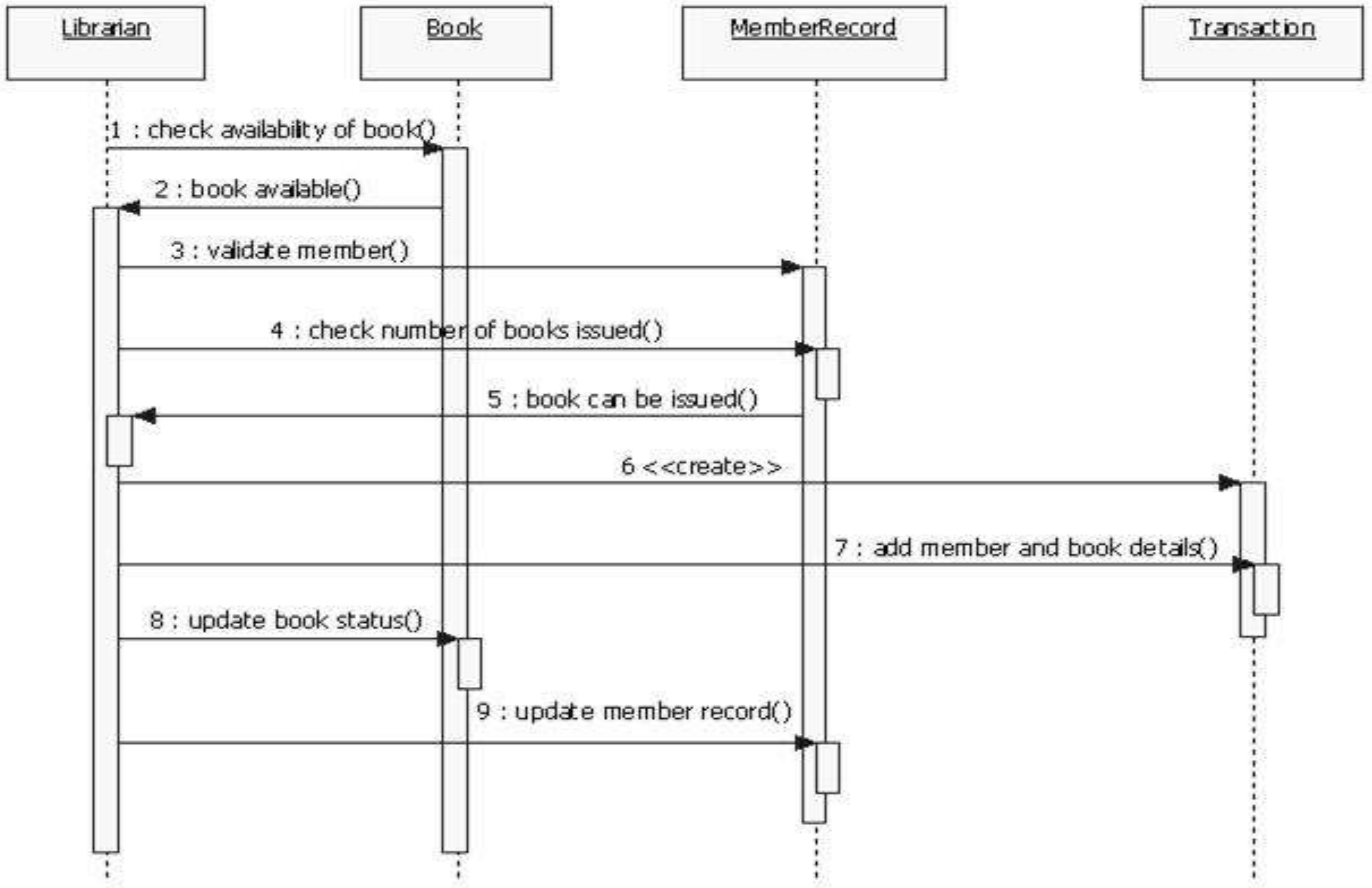

Sequence Diagram: Register for Courses / Create a Sched...
: Student
a form : MaintainSched...
: RegistrationController
: Student
a form : Maintain ScheduleForm
: Registration Controller
1: open
2: get course offerings

Focus of Control



Exercise: Sequence Diagram

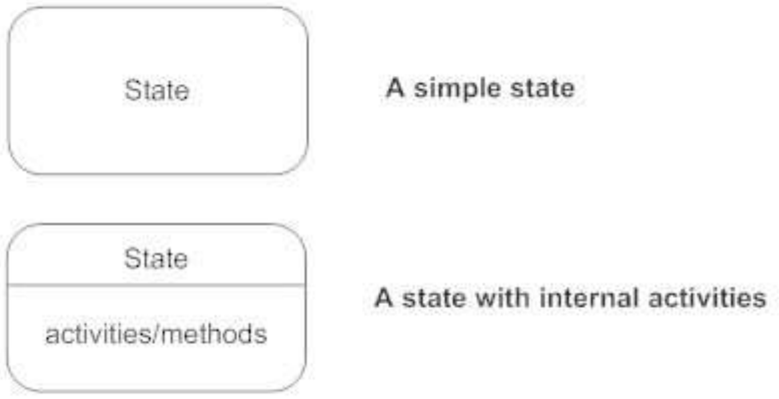



Librarian
Book
MemberRecord
Transaction
1 : check availability of book()
2 : book available()
3 : validate member()
4 : check number of books issued()
5 : book can be issued()
6 <<create>>
7 : add member and book details()
8 : update book status()
9 : update member record()

State Diagrams

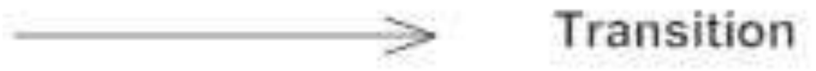
- State diagrams: similar in function to sequence diagrams, but with focus on the prerequisites for an operation, rather than the exact sequence of actions.

Basic State chart Diagram Symbols and Notations



States represent situations during the life of an object

Basic State chart Diagram Symbols and Notations

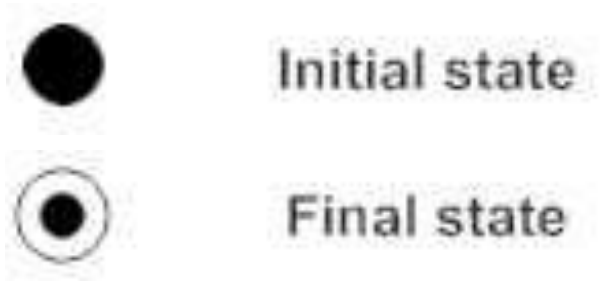


Transition

A solid arrow represents the path between different states of an object. Label the transition with the event that triggered it and the action that results from it. A state can have a transition that points back to itself.

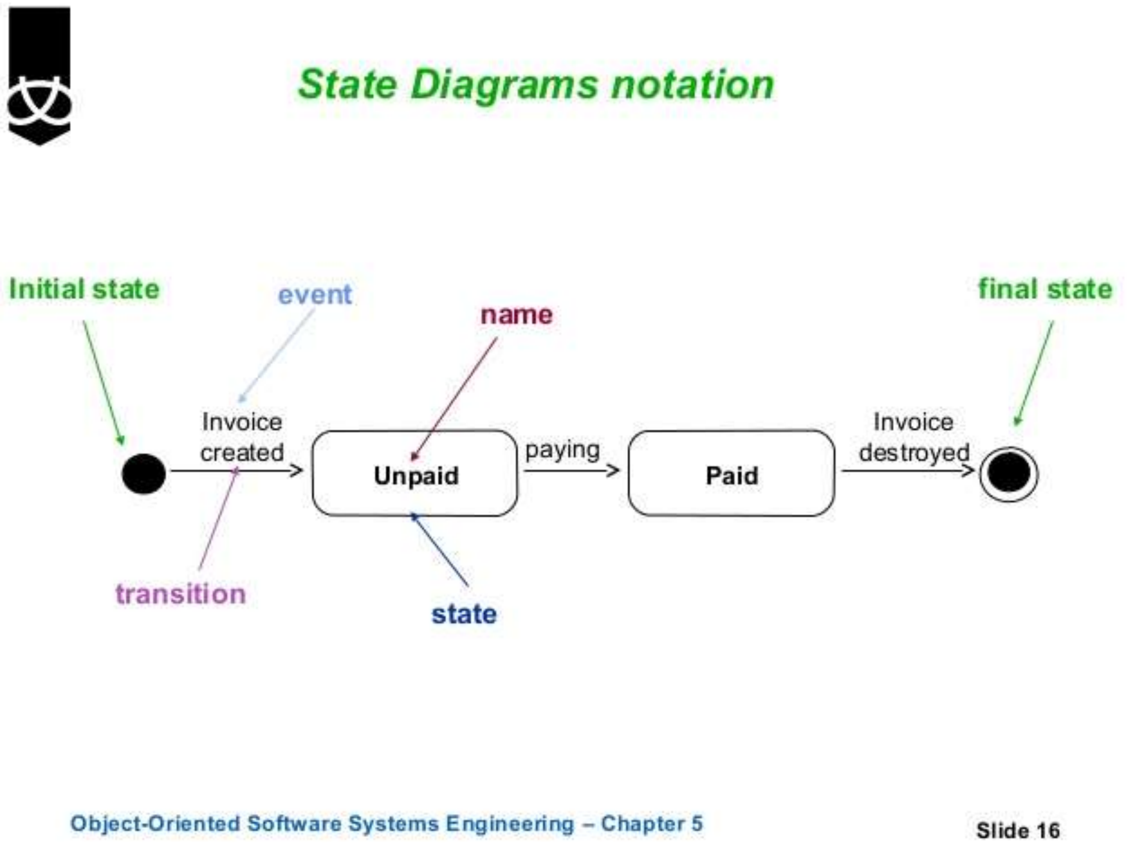
State Diagram

Initial State
A filled circle followed by an arrow represents the object's initial state.



Final State
An arrow pointing to a filled circle nested inside another circle represents the object's final state.

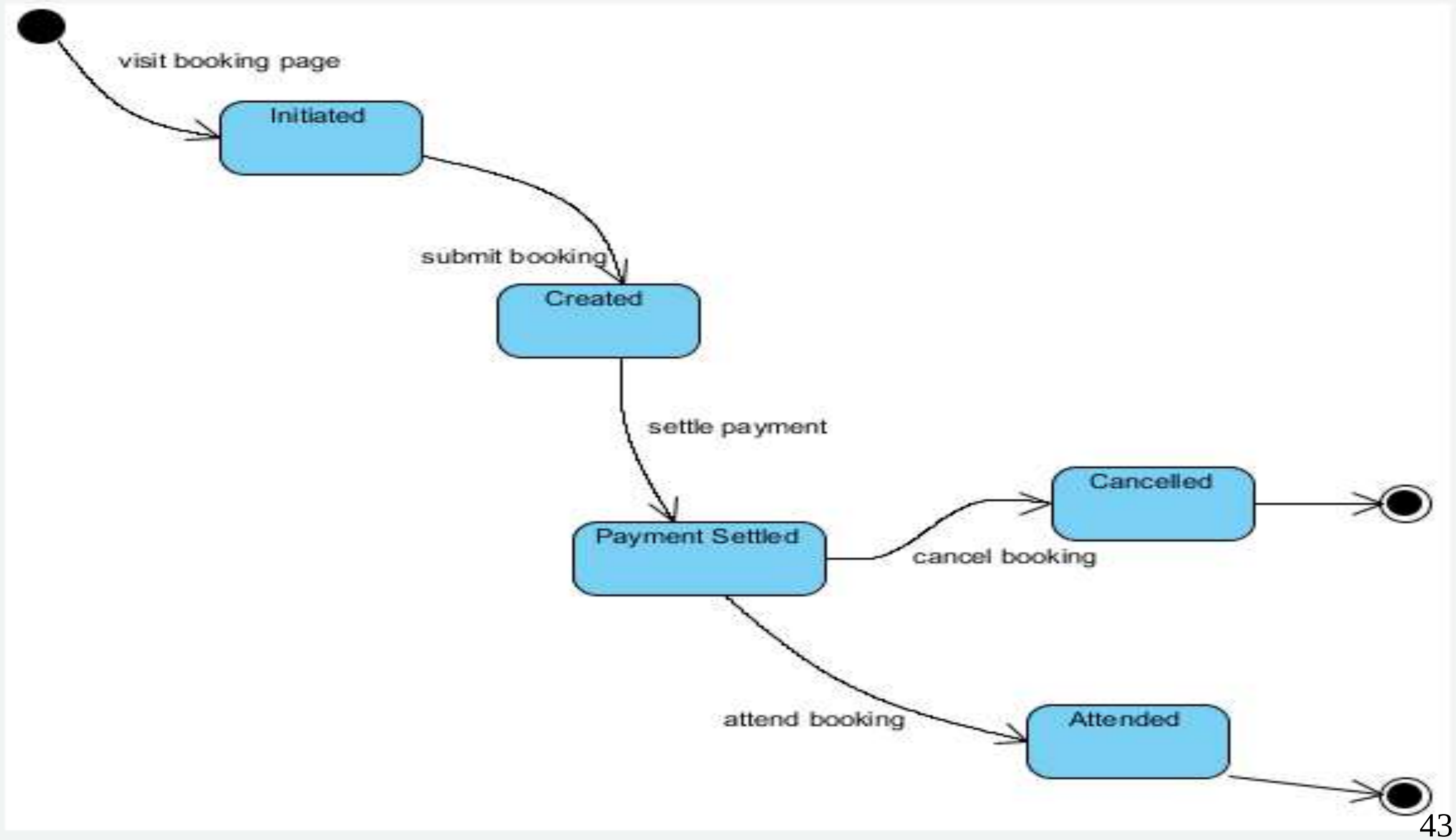
State Diagram


State Diagrams notation

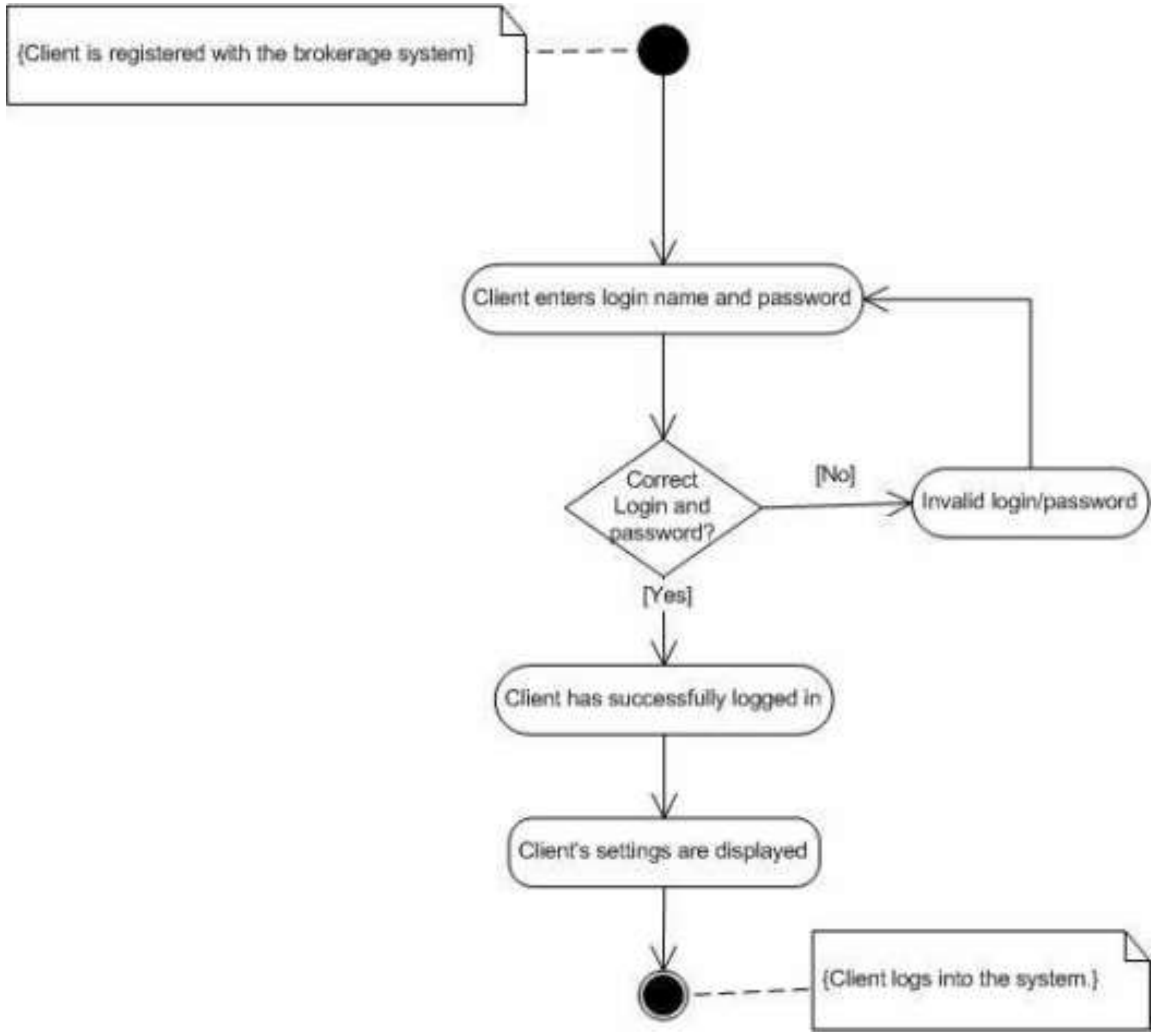
Object-Oriented Software Systems Engineering – Chapter 5

Slide 16

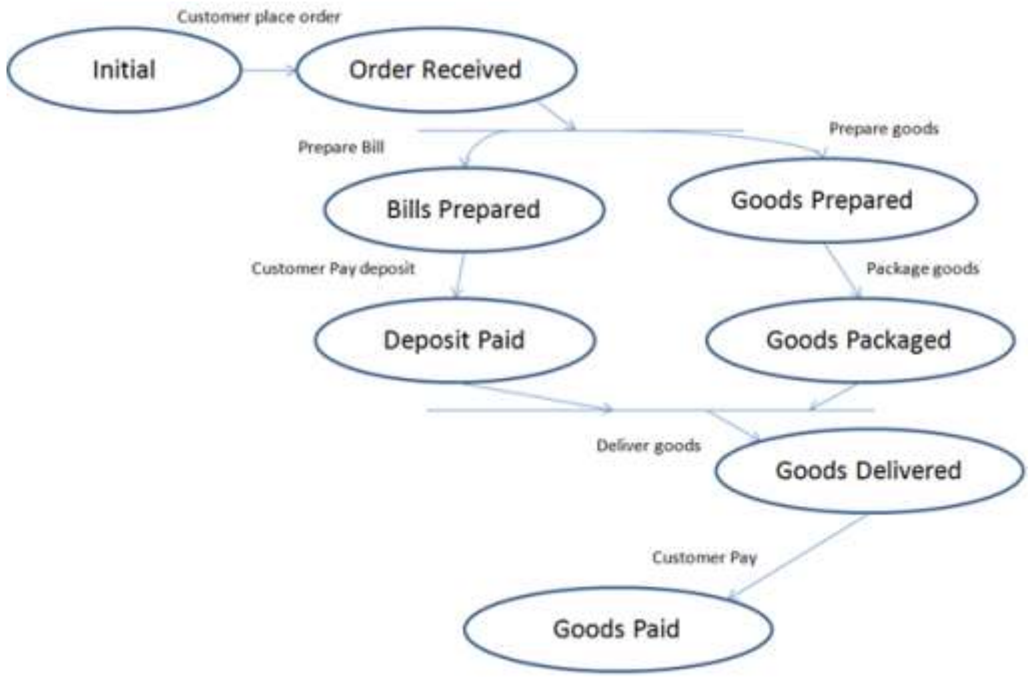
State Diagram



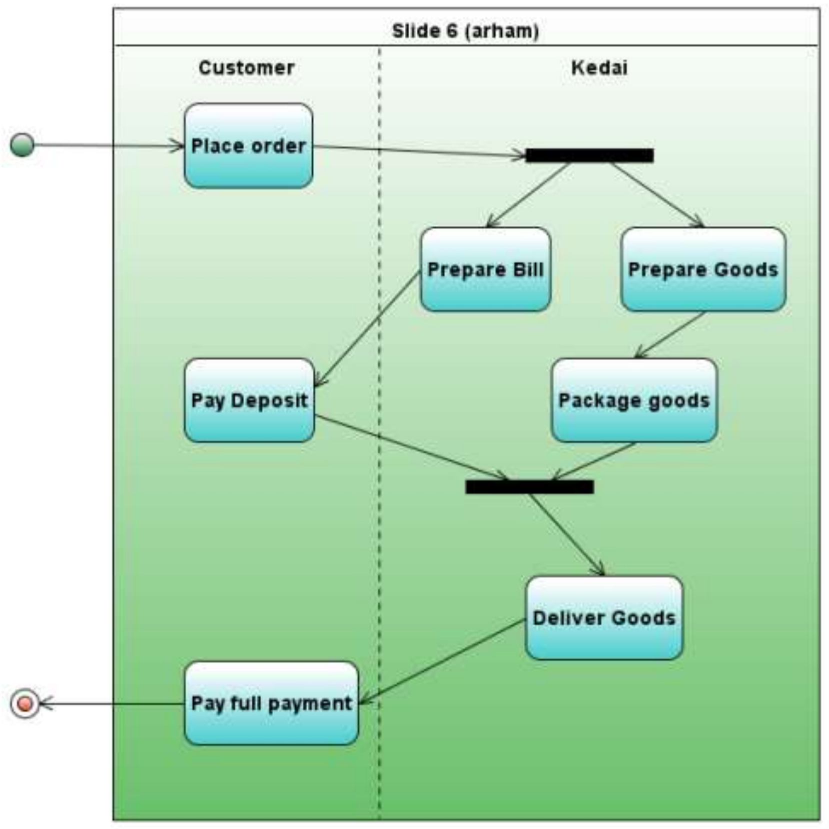
Activity Diagram


{Client is registered with the brokerage system}
Client enters login name and password
Correct Login and password?
[No]
Invalid login/password
[Yes]
Client has successfully logged in
Client's settings are displayed
{Client logs into the system.}

State Diagram



Activity Diagram


Slide 6 (arham)
Customer
Kedai
Place order
Prepare Bill
Prepare Goods
Pay Deposit
Package goods
Deliver Goods
Pay full payment