

Unified Modelling Language (UML)

What is UML?

- Unified Modeling Language (UML) is a standard graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems.
- Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the object-oriented software.
- UML was developed by the Object Management Group (www.omg.org) based on work from Booch, Rumbaugh, and Jacobson.
- The current version is UML 2.0.
- UML is NOT a methodology.
- UML is a notation for recording the results of
 - A requirements gathering / design process
 - Can be carried out using some methodology such as Iterative Model.
- UML uses mostly graphical notations to express the design of software projects.
- UML is independent of implementation language.

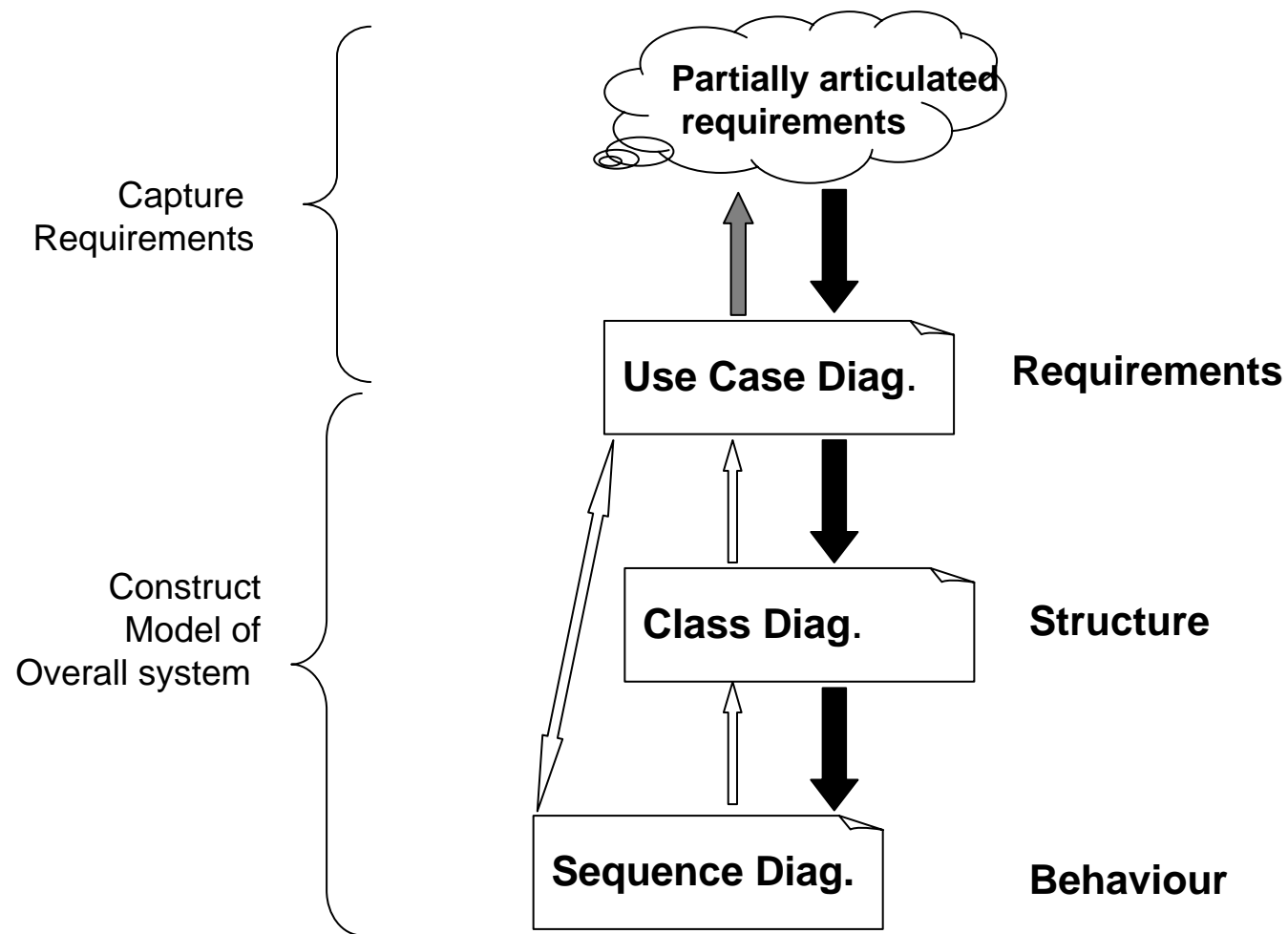
Why use UML?

- Standardized graphical notation for specifying, visualizing, constructing, and documenting software systems
- Language can be used from general initial design to very specific detailed design
- Increase understanding and communication of product to customers and developers
- Support for UML in many CASE tools today (e.g. IBM Rational Rose)

UML Diagrams

- **Class Diagrams**
- **Use Case Diagrams**
- **Interaction Diagrams**
 - **Sequence Diagrams**
 - **Collaboration Diagrams**
- **State Transition Diagrams**
- **Statechart Diagrams**
- **Activity Diagrams**
- **Implementation Diagrams**
 - **Component Diagrams**
 - **Deployment Diagrams**

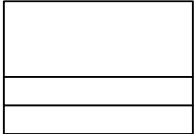


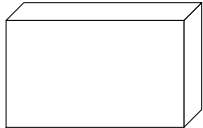
Minimal Development Process






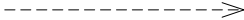
These diagram types, and the process can be used both for

- Designing a new system
- Understanding an existing system


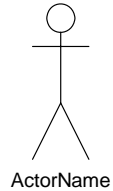

Structural Modeling: Core Elements

Construct	Description	Syntax
class	a description of a set of objects that share the same attributes, operations, methods, relationships and semantics.	
interface	a named set of operations that characterize the behavior of an element.	
component	a modular, replaceable and significant part of a system that packages implementation and exposes a set of interfaces.	
node	a run-time physical object that represents a computational resource.	




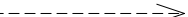
Structural Modeling: Core Relationships

Construct	Description	Syntax
association	a relationship between two or more classifiers that involves connections among their instances.	
aggregation	A special form of association that specifies a whole-part relationship between the aggregate (whole) and the component part.	
generalization	a taxonomic relationship between a more general and a more specific element.	
dependency	a relationship between two modeling elements, in which a change to one modeling element (the independent element) will affect the other modeling element (the dependent element).	

Use Case Modeling: Core Elements

Construct	Description	Syntax
use case	A sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.	
actor	A coherent set of roles that users of use cases play when interacting with these use cases.	
system boundary	Represents the boundary between the physical system and the actors who interact with the physical system.	

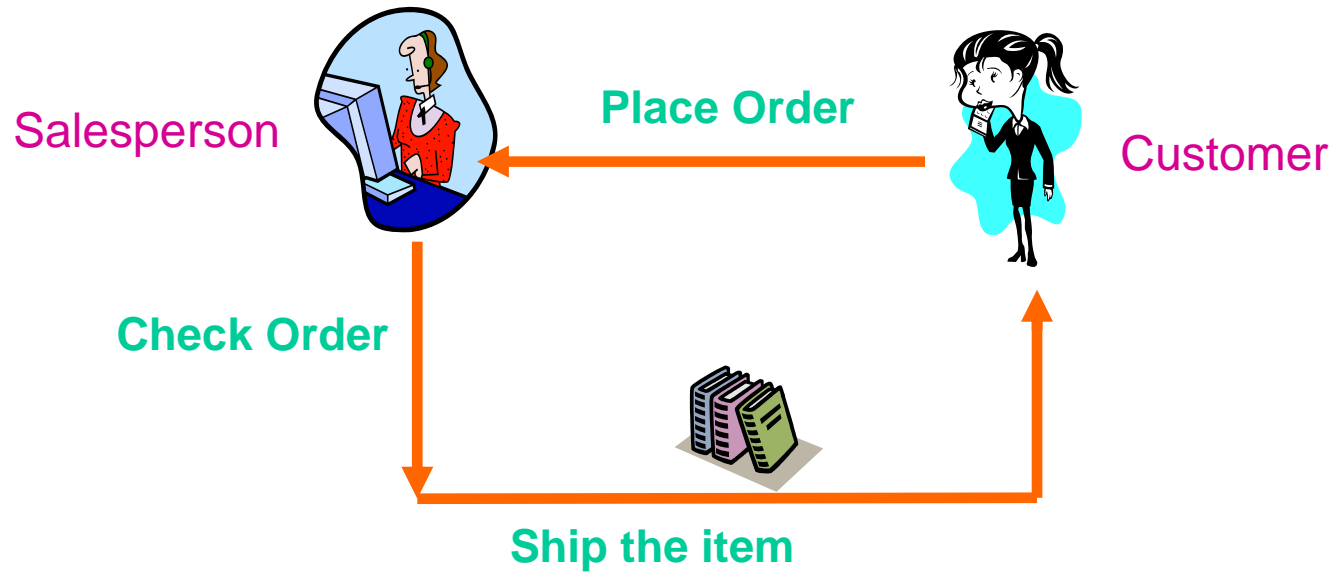
Use Case Modeling: Core Relationships

Construct	Description	Syntax
association	The participation of an actor in a use case. i.e., instance of an actor and instances of a use case communicate with each other.	
generalization	A taxonomic relationship between a more general use case and a more specific use case.	
extend	A relationship from an <i>extension</i> use case to a <i>base</i> use case, specifying how the behavior for the extension use case can be inserted into the behavior defined for the base use case.	<<extend>> 
include	An relationship from a <i>base</i> use case to an <i>inclusion</i> use case, specifying how the behavior for the inclusion use case is inserted into the behavior defined for the base use case.	<<include>> 

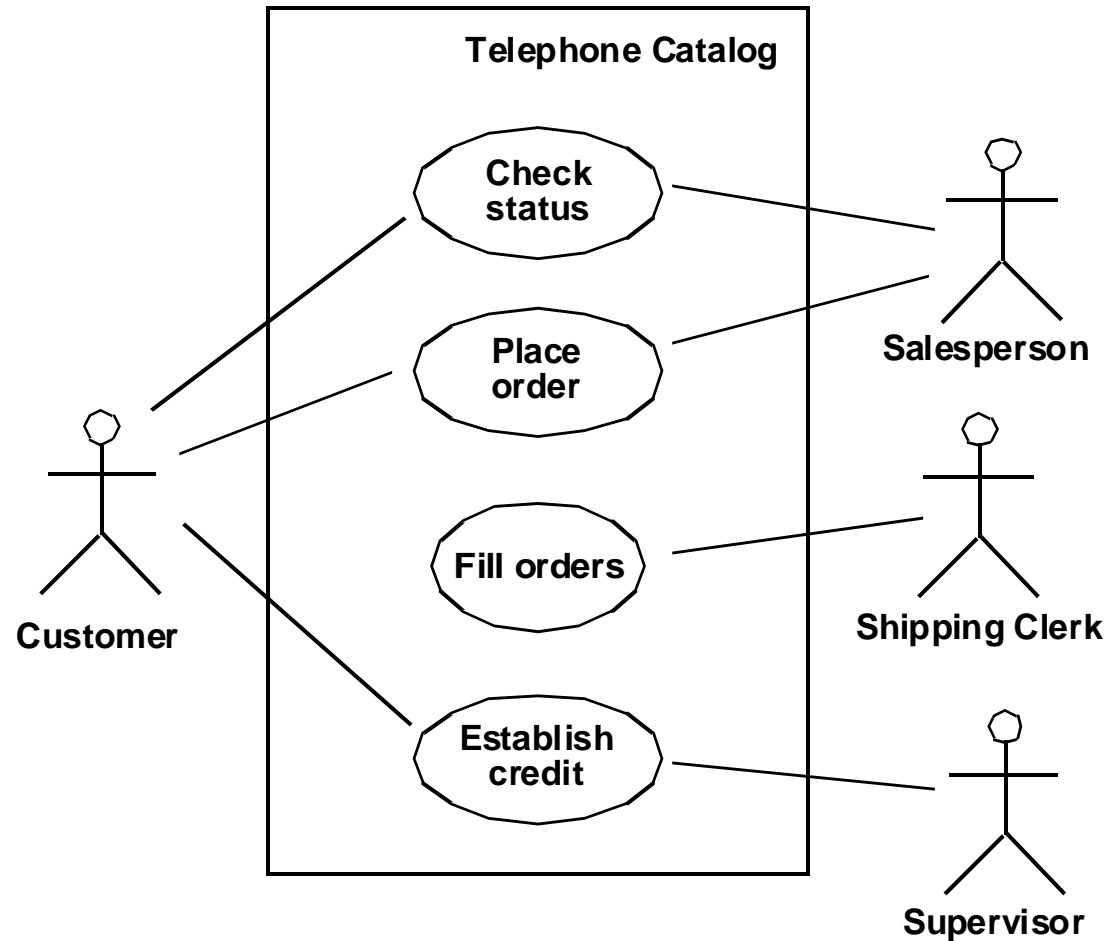
Example-1:

Sales System

Business process

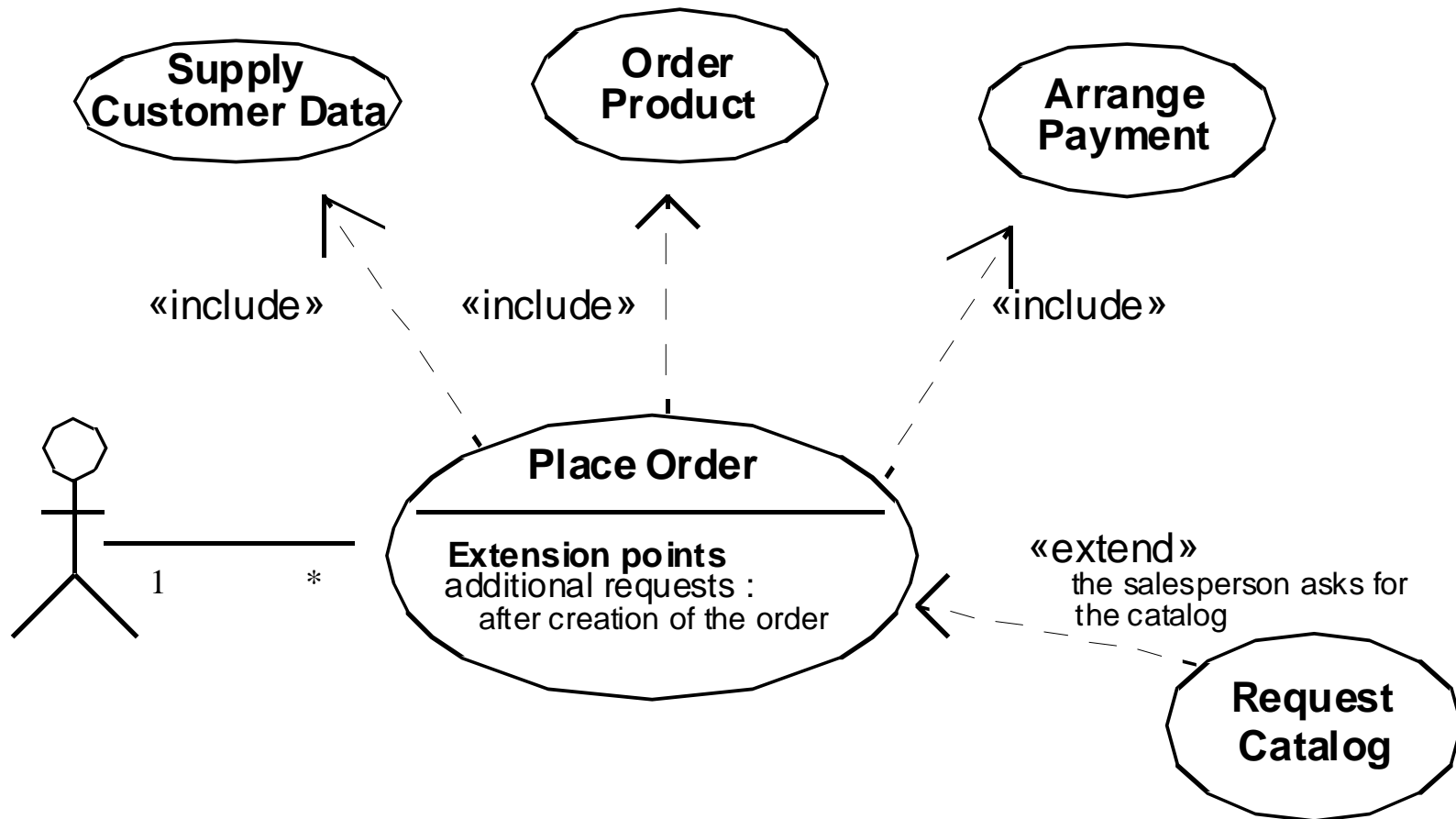


Use Case Diagram

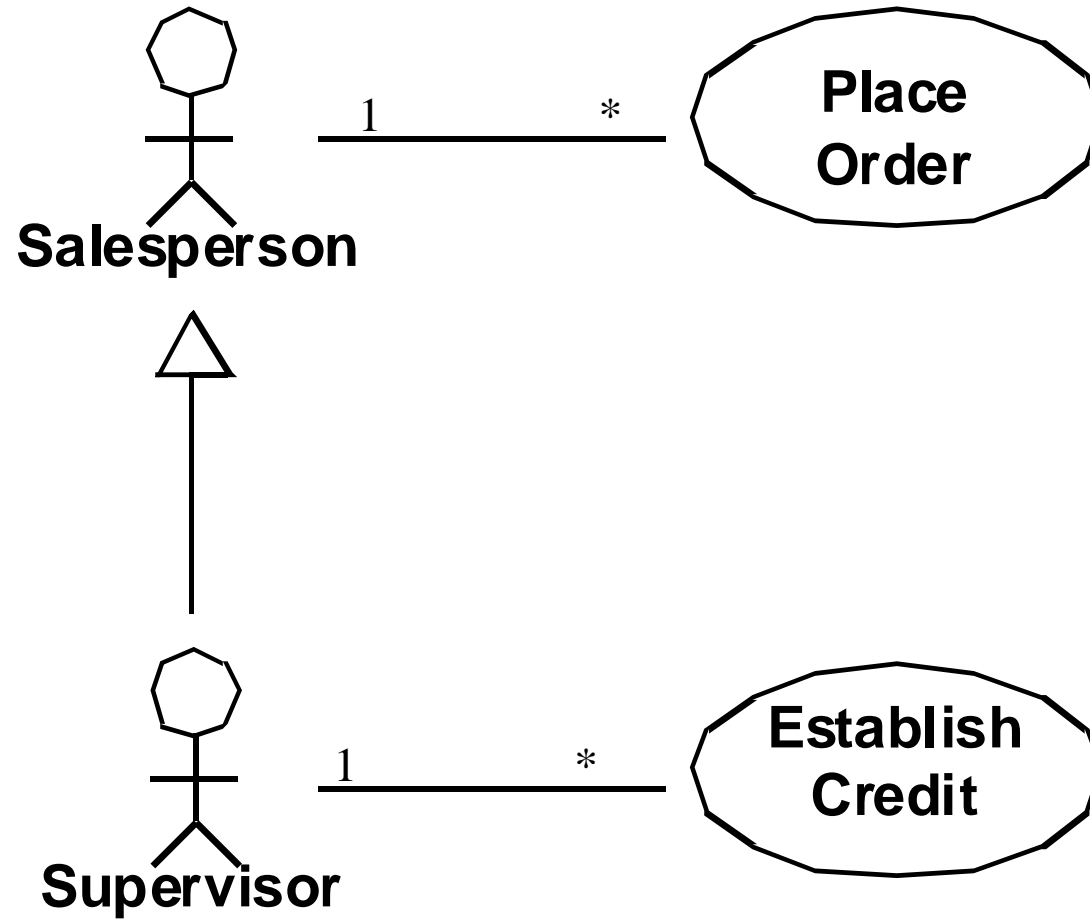


- Use cases capture the requirements.

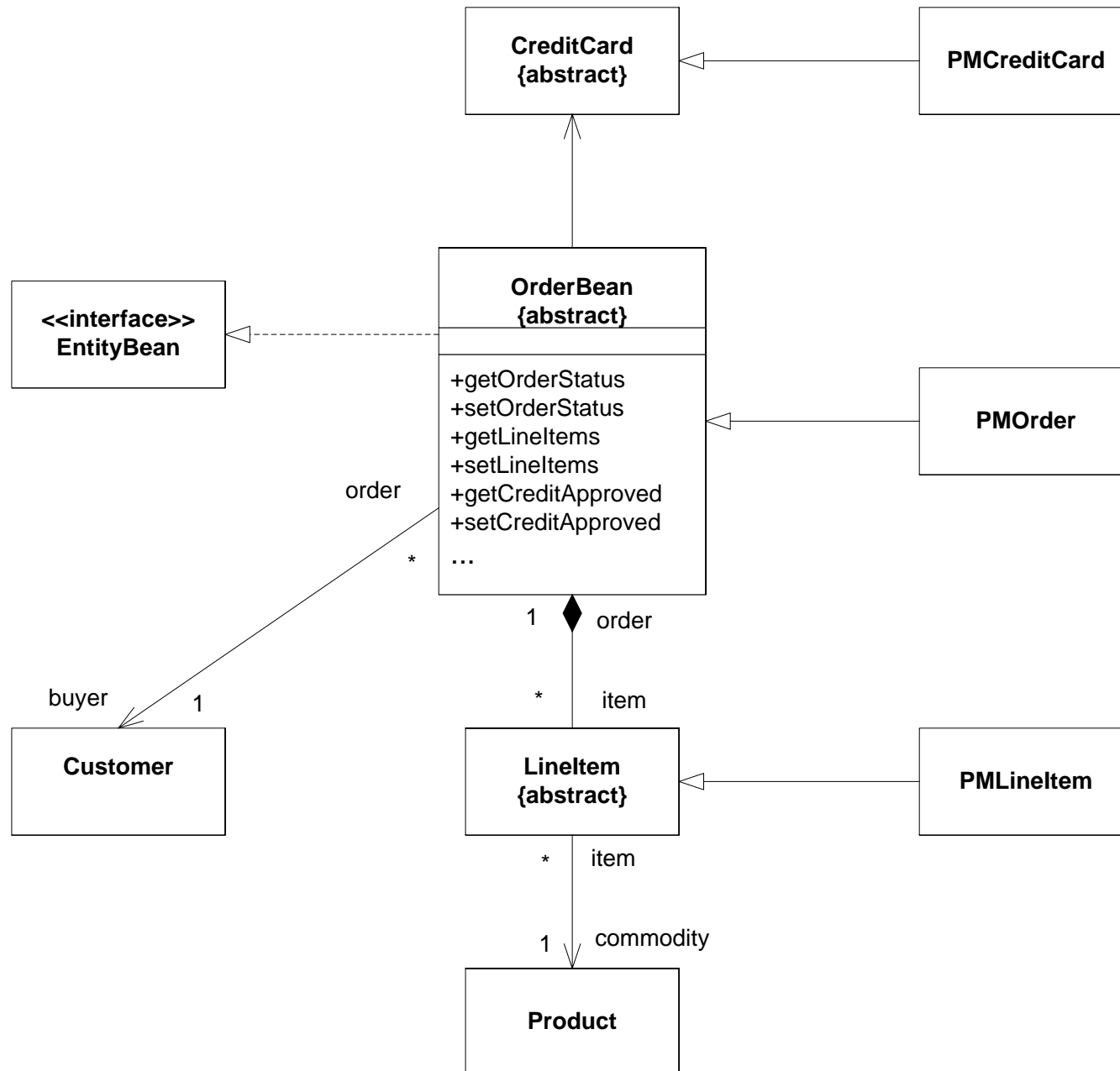
Use Case Relationships



Actor Relationships



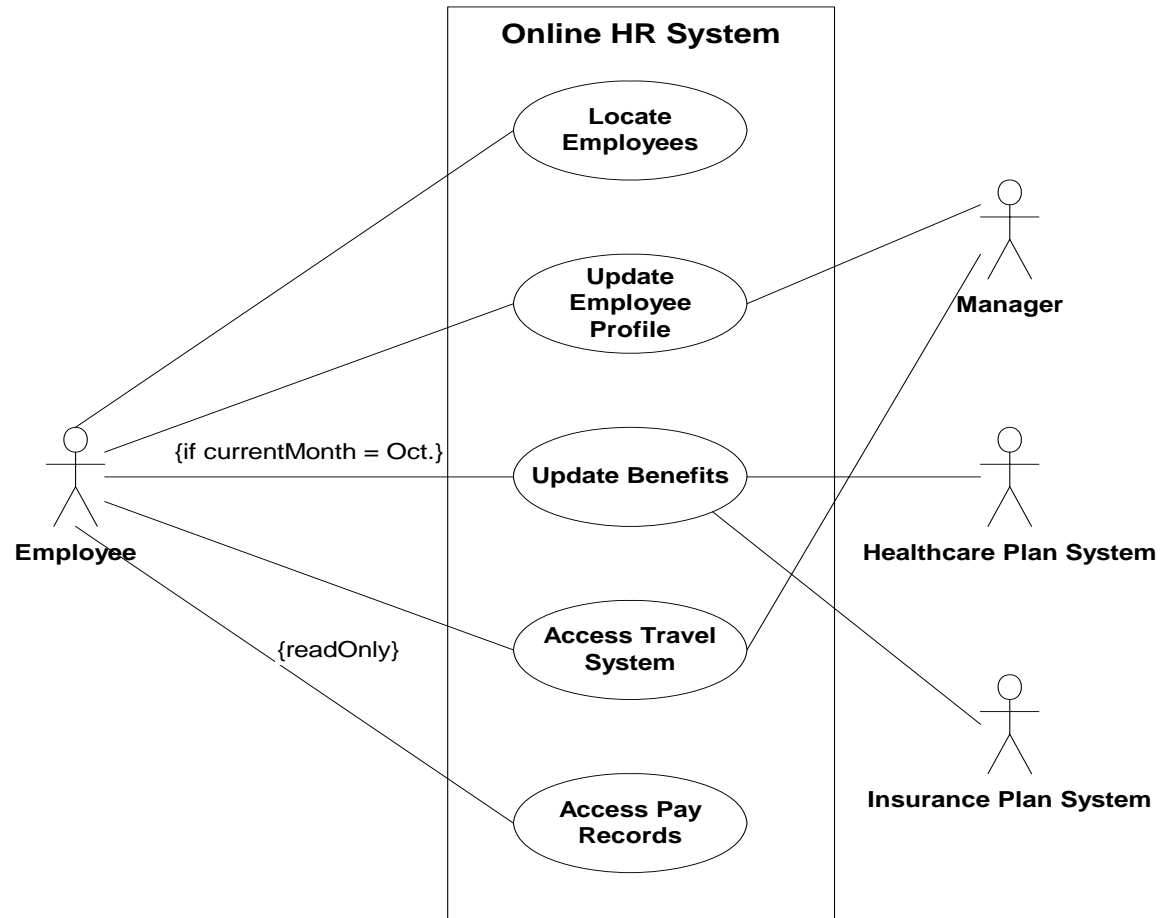
Class Diagram



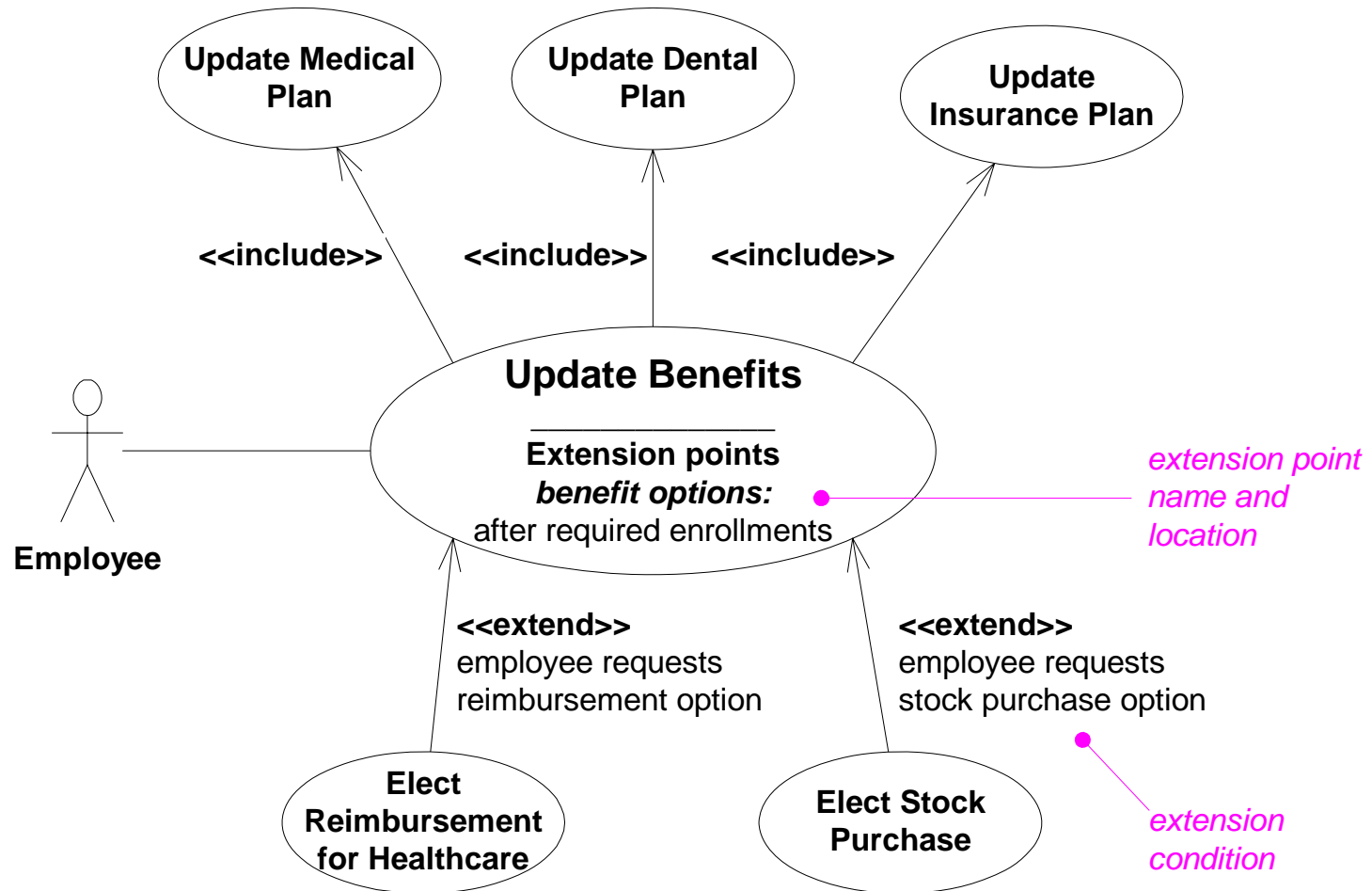
Example-2:

Online Human Resources (HR) System

Use Case Diagram



Use Case Relationships



Use Case Description: “Update benefits”

- **Actors:** employee, employee account db, healthcare plan system, insurance plan system
- **Preconditions:**
 - Employee has logged on to the system and selected ‘update benefits’ option
- **Basic use case:**
 - System retrieves employee account from employee account db
 - System asks employee to select medical plan type; **include** Update Medical Plan.
 - System asks employee to select dental plan type; **include** Update Dental Plan.

Example-3:

Library System

Description

- A computer system for a university library; some “facts”
- The library has members
 - Staff
 - Students
- The library contains items
 - Books
 - possibly several copies of each book
 - Some books are for short-term loan only – only students can do short-term loans
 - Journals
 - Only staff may borrow journals
- The functionalities include
 - Keeping track of when items are borrowed/returned
 - Allow users to search for a book and check availability

Some queries

The library has members –

- Staff
- Students

← **Can someone be both?**

• The library contains

– Books;

- possibly several copies of each book

- Some books are for short-term loan only – only students can do short-term loans

Do we mean copies?

– Journal

- Only staff may borrow journals

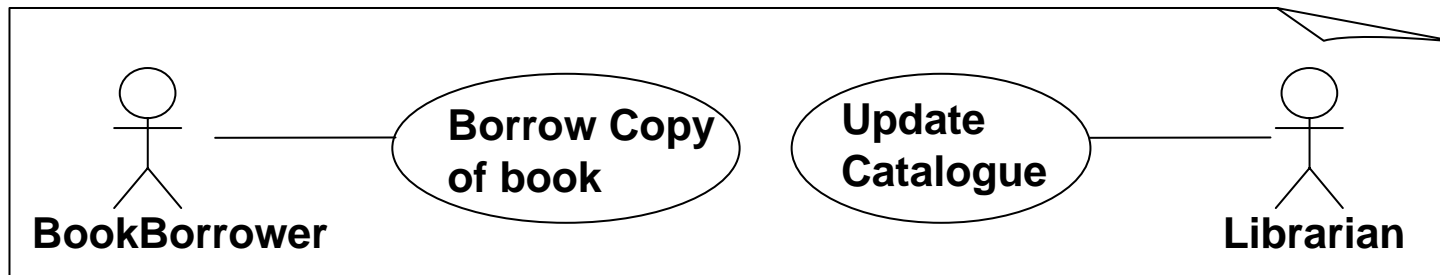
• The functionalities include

– Keeping track of when items are borrowed/returned

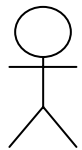
– Allow users to search for a book and check availability

← **Is “user” something different from “member?”**
Henceforth use “user”

Requirements - Simple Use Cases

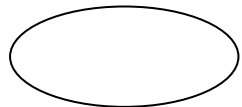


An Actor



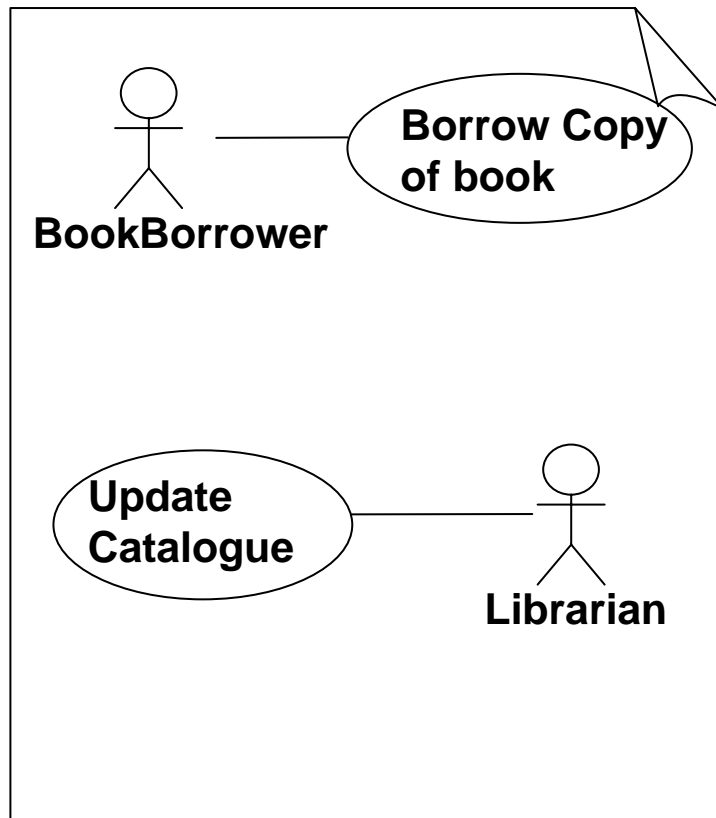
- a user of the system, in a particular role
- “BookBorrower” is a role played by library members
- Usually a person, but could be another system
 - An entity outside the system being modelled
 - In web services – usually another web service

A Use Case



- A task that an actor needs to perform with the help of the system
- “Borrow Copy of Book” describes the successful outcome – there will be variants for the exception cases

Simple Use Cases



Use Case Documentation

Use Case: Borrow copy of book.

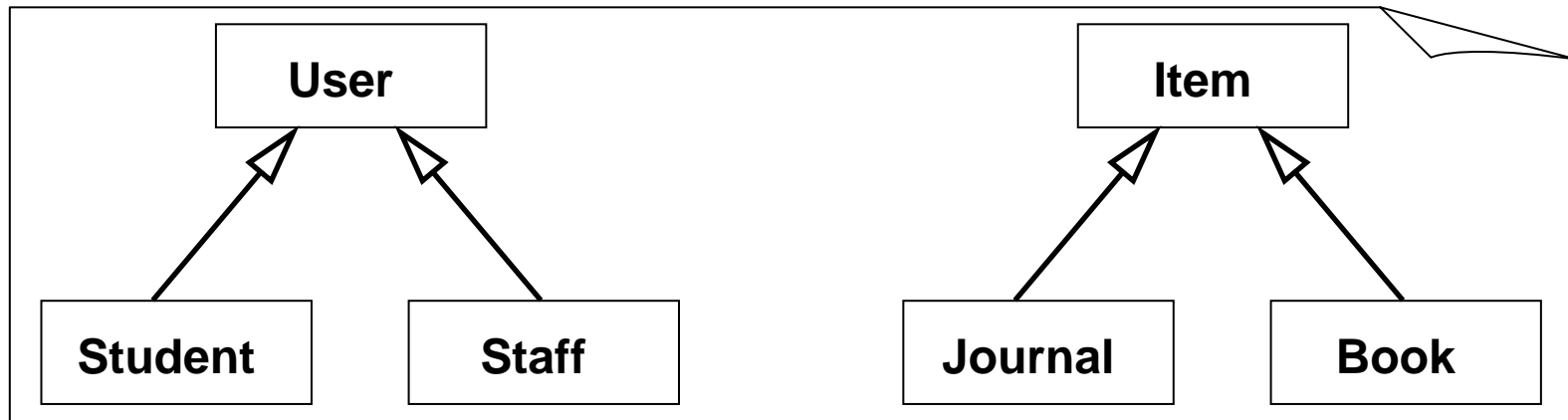
- A bookBorrower presents a book.
- The system checks that the potential borrower is a member of the library,
- and that s/he does not already have the maximum permitted number of books on loan.
- The maximum is 6 unless the member is is a staff member, in which case it is 12.
- If both ckecks succeed, the system records that this library member has this copy of the book on loan.
- Otherwise it refuses the loan.

- Not a very rich diagram – later on we will see richer diagrams
- Needs more explanation – associated documentation

Structure - Class Design

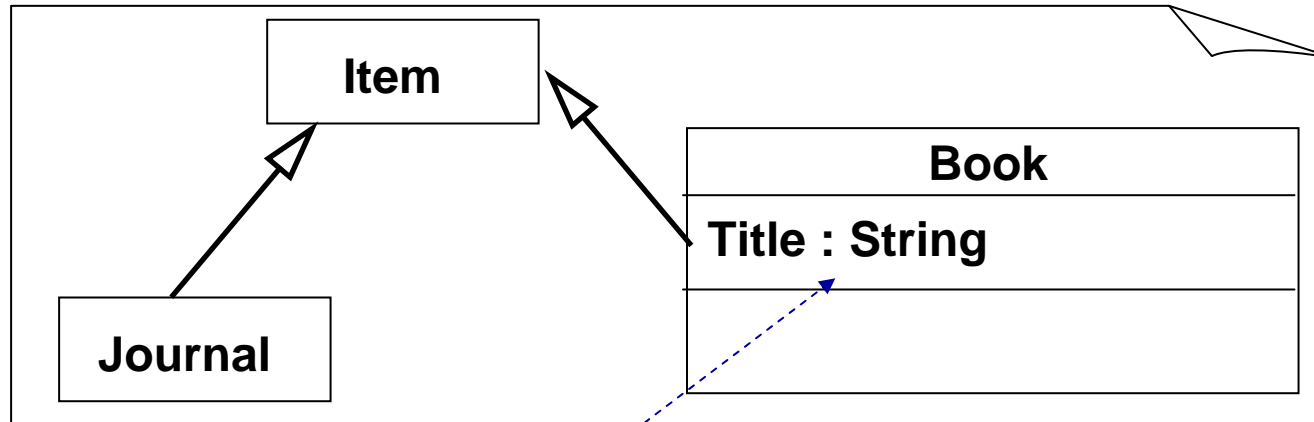
- UML is for Object-Oriented Design
 - The state of the system is a collection of Objects
 - Each Object is an instance of a Class
 - The structure of the system is characterised by the class and their inter-relationships
- Key is identifying the classes
 - kinds of things manipulated within the system
- Standard Technique - Noun identification
 - Take a concise description of the system
 - Identify the words/phrases that denote *things*

Class Diagram



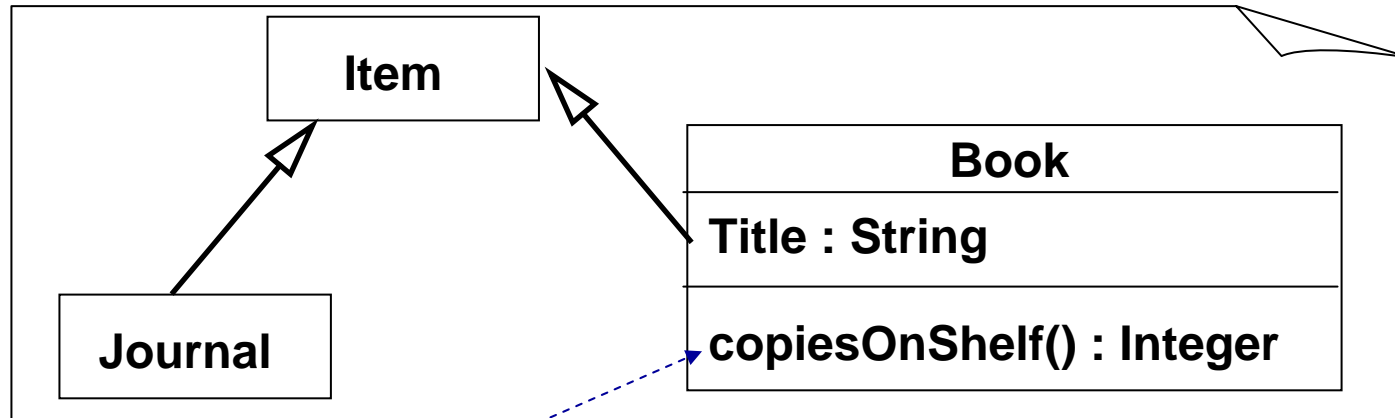
- A box for each Class
- There are relationships between classes
- Specialisation/Generalisation
 - User is a generalisation of both
 - **Student**
 - **Staff**
 - Any instance of Student is also an instance of User

Class Attributes and Operations



- **Attribute – Information associated with each instance of the class**
- **Each attribute will have a type**

Class Attributes and Operations



- **An operation – something that instances of this class can do, which can be externally invoked**
- **Equivalent to a Java Method**

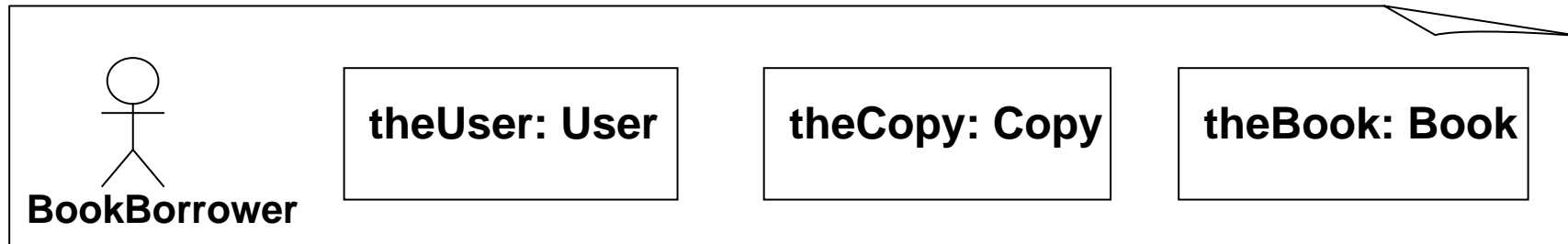
- **UML terminology is message-passing - An object's operation is invoked by another object sending it a message**

Behaviour - Sequence Diagrams

- To document how objects work together to achieve an overall function of the system
- Each sequence diagram shows an example “walk-through”
- Typically for a use case

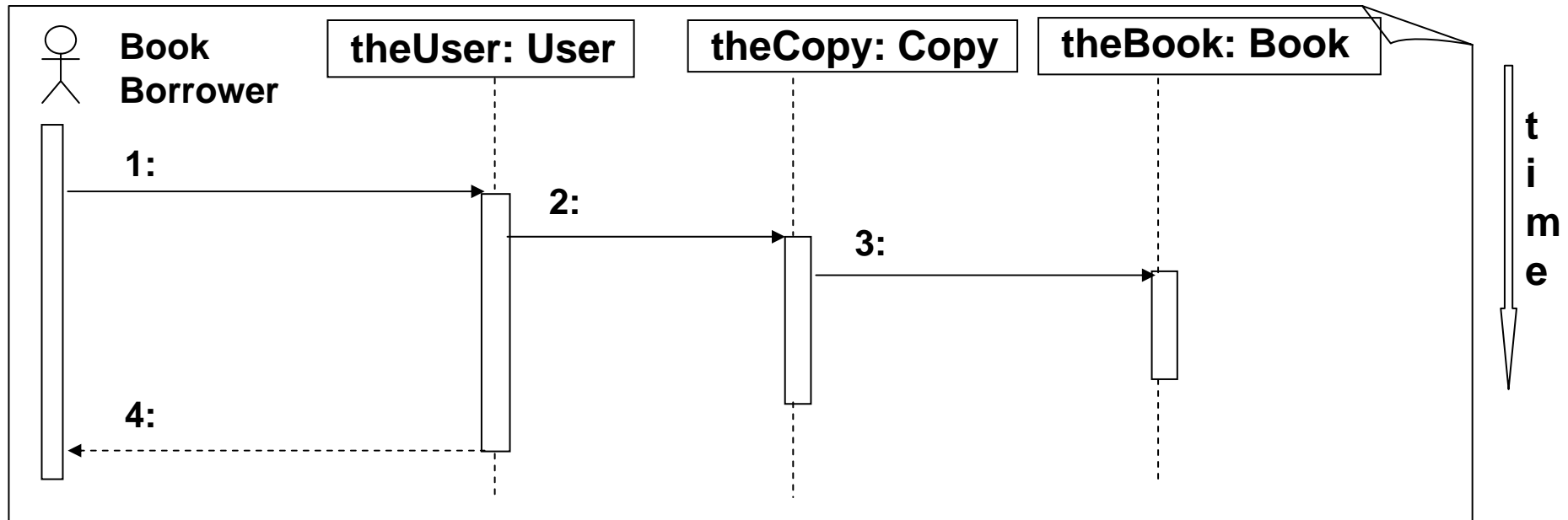
Sequence Diagram - participants

- For Use Case : Borrow copy of book



- Identify the participants
 - External Actor from the use case
 - Objects within the system
- This is an instance diagram –
Name each object, and identify its class
 - theUser – object which is the internal representation of the user playing the BookBorrower role
 - “theUser : User” seems redundant, but could have several User participants

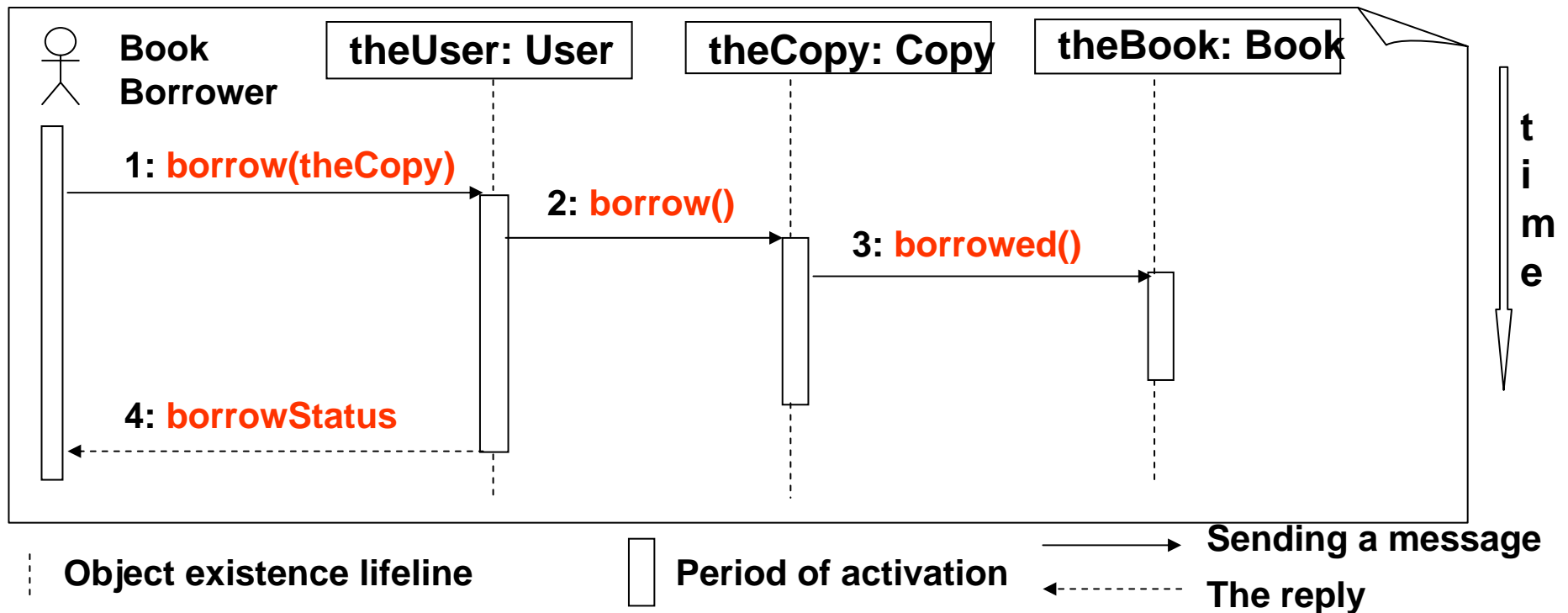
Sequence Diagram – lifelines



1. **BookBorrower** invokes the use case – activating the corresponding **User** object, **theUser**
2. **theUser** tells **theCopy** object that it is now borrowed – it changes its state
3. **theCopy** tells **theBook** that it has been borrowed – its reduces its count of available copies
4. **TheUser** confirms success to **BookBorrower**

The replies to 2 and 3 not shown because not particularly important, whereas the confirmation to external user is.

Sequence Diagram – operations



Associate with the message line –

The information communicated

Usually the operation invoked and possibly its parameters

Example-4:

University Registration System

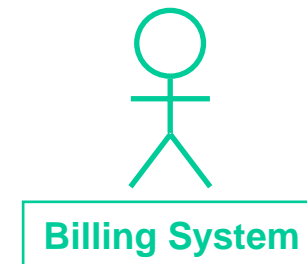
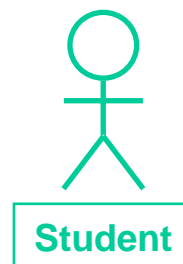
Description

Suppose that Istanbul Technical University wants to computerize its registration system:

- The Registrar sets up the curriculum for a semester
 - One course may have multiple course offerings
- Students select four (4) primary courses and two (2) alternate courses
- Once a student registers for a semester, the billing system is notified so the student may be billed for the semester
- Students may use the system to add/drop courses for a period of time after registration
- Professors use the system to set their preferred course offerings and receive their course offering rosters after students register
- Users of the registration system are assigned passwords which are used at logon validation

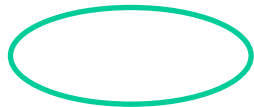
Actors in Use Case Diagram

- An **actor** is someone or some thing that must interact with the system under development

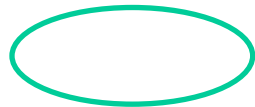


Use Cases in Use Case Diagram

- A **use case** is a pattern of behavior the system exhibits
 - Each use case is a sequence of related transactions performed by an actor and the system in a dialogue
- Actors are examined to determine their needs
 - Registrar -- maintain the curriculum
 - Professor – set course offerings and request roster
 - Student -- register for courses
 - Billing System -- receive billing information from registration



Maintain Curriculum



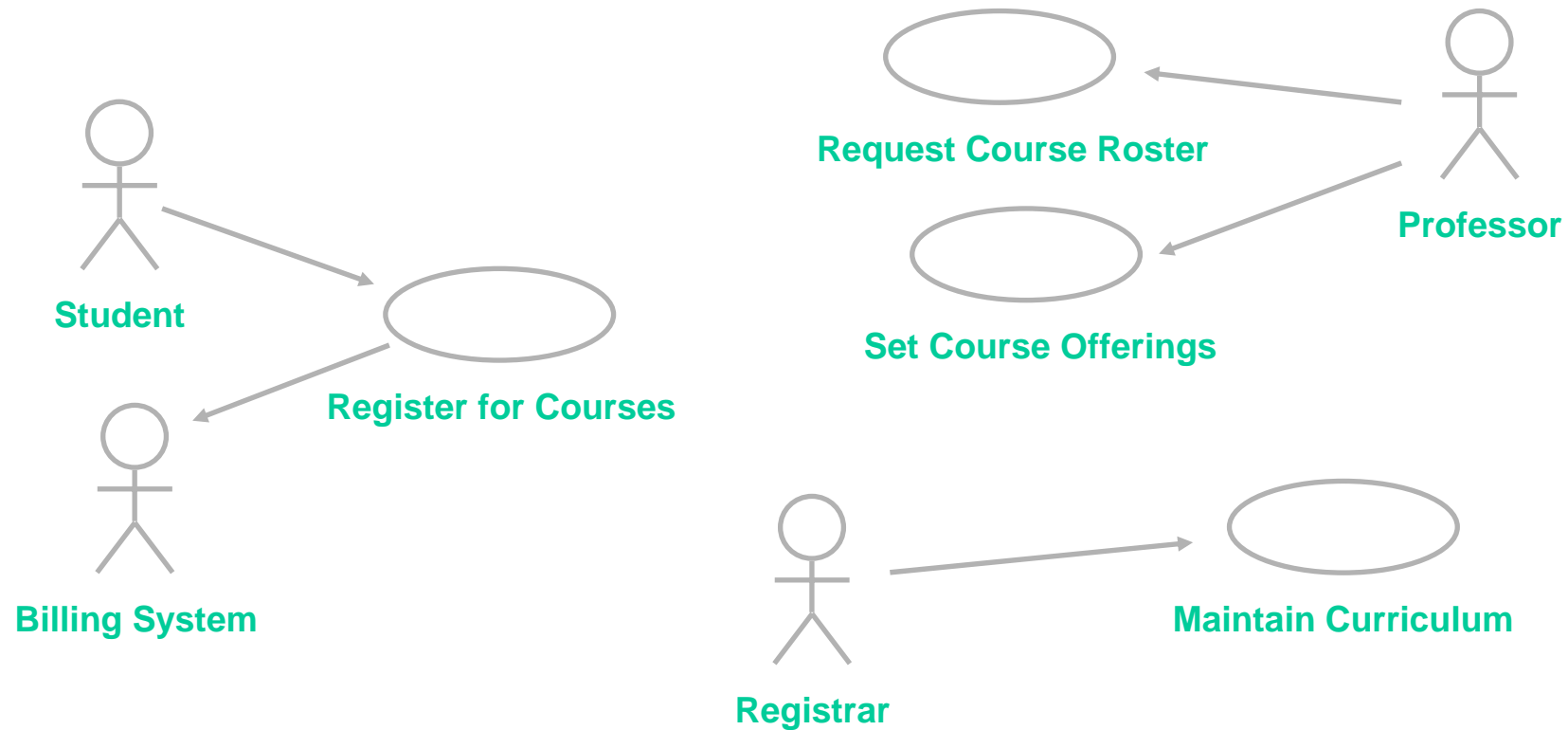
Request Course Roster



Register for Courses

Use Case Diagram

- Use case diagrams are created to visualize the relationships between actors and use cases



Flow of Events

Flow of Events for Maintaining Curriculum



- This use case begins when the Registrar logs onto the Registration System and enters his/her password.
- The system verifies that the password is valid and prompts the Registrar to select the current semester or a future semester.
- The Registrar enters the desired semester.

Flow of Events for Setting Course Offerings

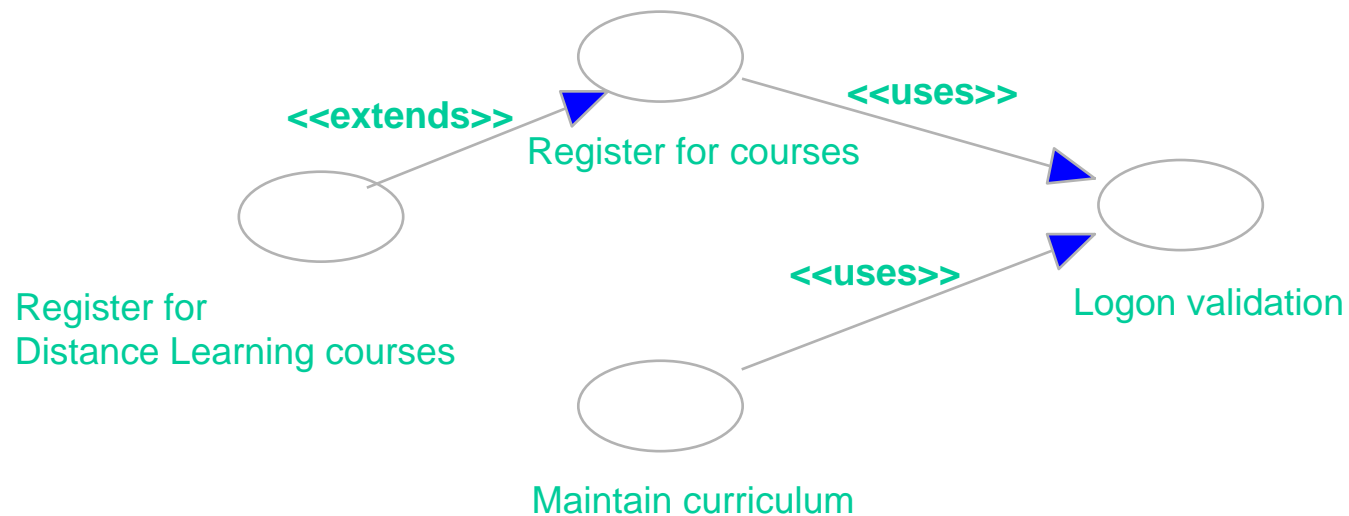


- The system prompts the professor to select the desired activity: ADD, DELETE, REVIEW, or QUIT.
- If the activity selected is ADD: *Add a Course* subflow is performed.
- If the activity selected is DELETE: *Delete a Course* subflow is performed.
- If the activity selected is REVIEW: *Review Curriculum* subflow is performed.
- If the activity selected is QUIT, the use case ends.

Uses and Extends Relationships in Use Case Diagram

A **uses** relationship shows behavior common to one or more use cases

An **extends** relationship shows optional behavior

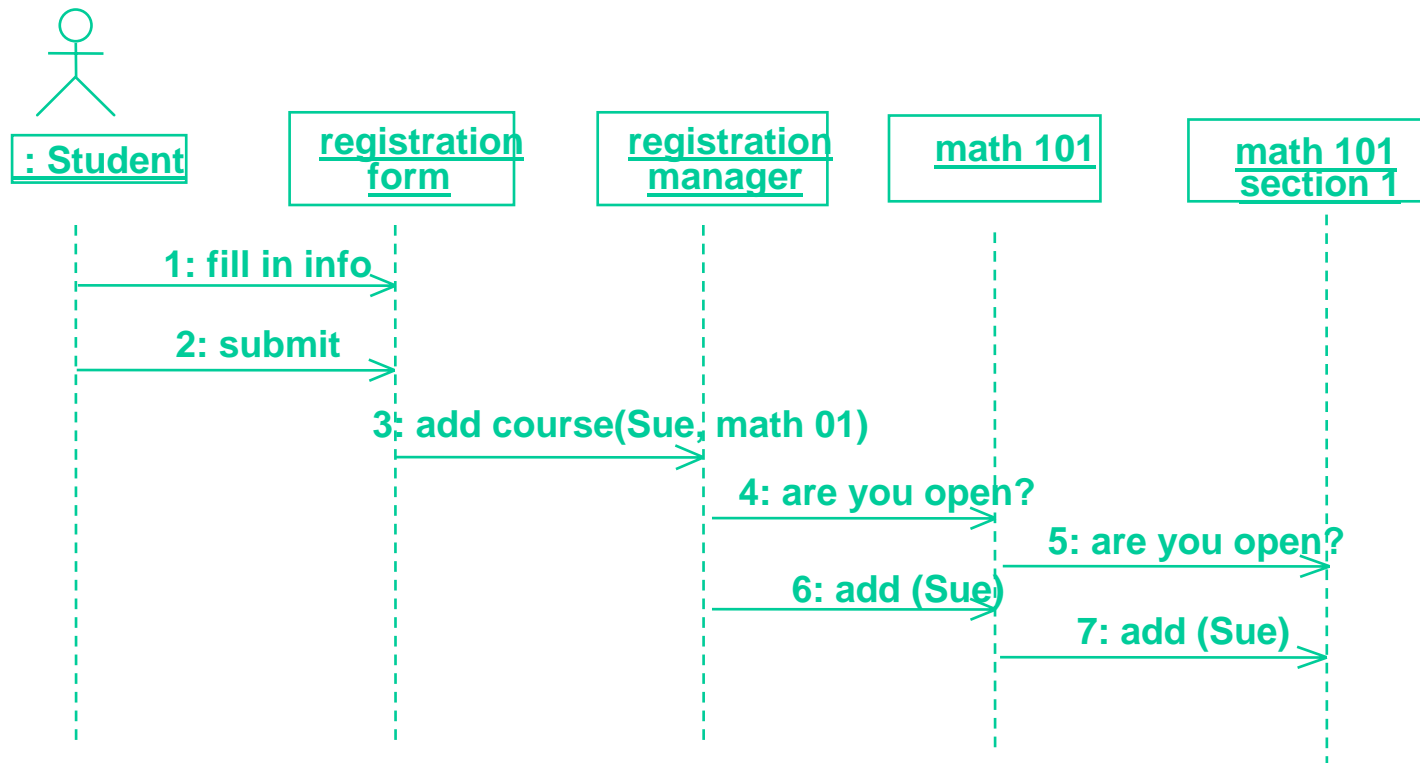


Interaction Diagrams

- A use case diagram presents an outside view of the system.
- Then, how about the inside view of the system?
- Interaction diagrams describe how use cases are realized as interactions among societies of objects, including the messages that may be dispatched among them. They address the dynamic view of the system.
- Two types of interaction diagrams
 - Sequence diagrams
 - Collaboration diagrams

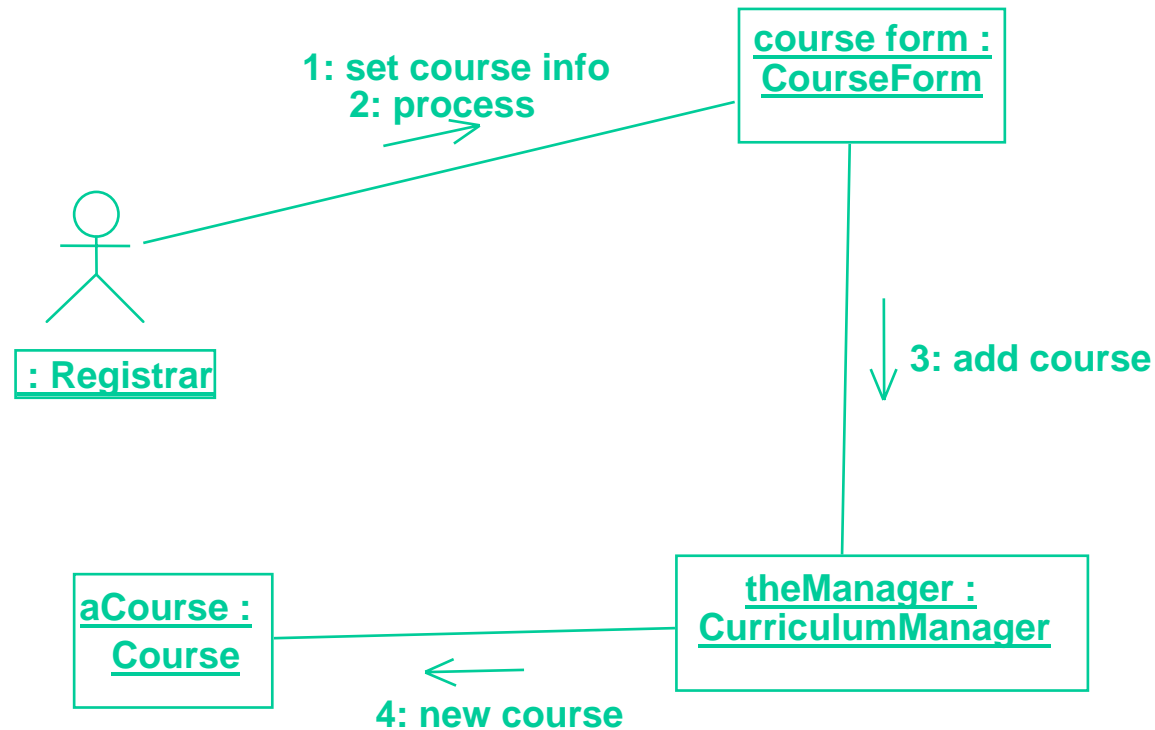
Sequence Diagram

- A sequence diagram displays object interactions arranged in a time sequence

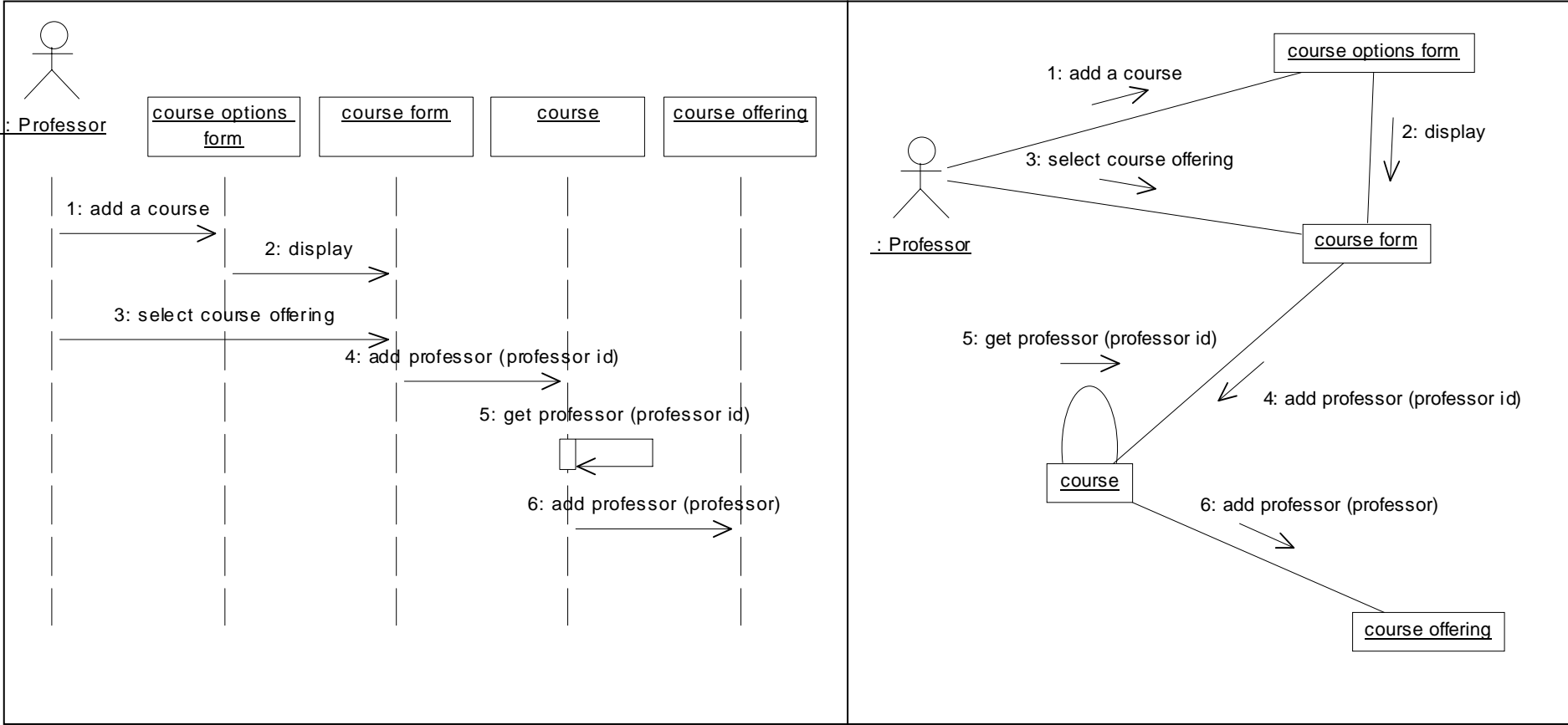


Collaboration Diagram

- Displays object interactions organized around objects and their **direct** links to one another.
- Emphasizes the structural organization of objects that send and receive messages.



Sequence and Collaboration Diagrams



Class Diagrams

- A class diagram shows the existence of classes and their relationships in the logical view of a system
- UML modeling elements in class diagrams
 - Classes and their structure and behavior
 - Association, aggregation, dependency, and inheritance relationships
 - Multiplicity and navigation indicators
 - Role names
- A **class** is a collection of objects with common structure, common behavior, common relationships and common semantics
- Some classes are shown through the objects in sequence and collaboration diagram
- A class is drawn as a rectangle with three compartments
- Classes should be named using the vocabulary of the domain
 - Naming standards should be created
 - e.g., all classes are singular nouns starting with a capital letter

Classes: Operations and Attributes

Operations

- The behavior of a class is represented by its operations
- Operations may be found by examining interaction diagrams

Attributes

- The structure of a class is represented by its attributes
- Attributes may be found by examining class definitions, the problem requirements, and by applying domain knowledge

Classes: Operations and Attributes

RegistrationForm

ScheduleAlgorithm

RegistrationManager
addStudent(Course, StudentInfo)

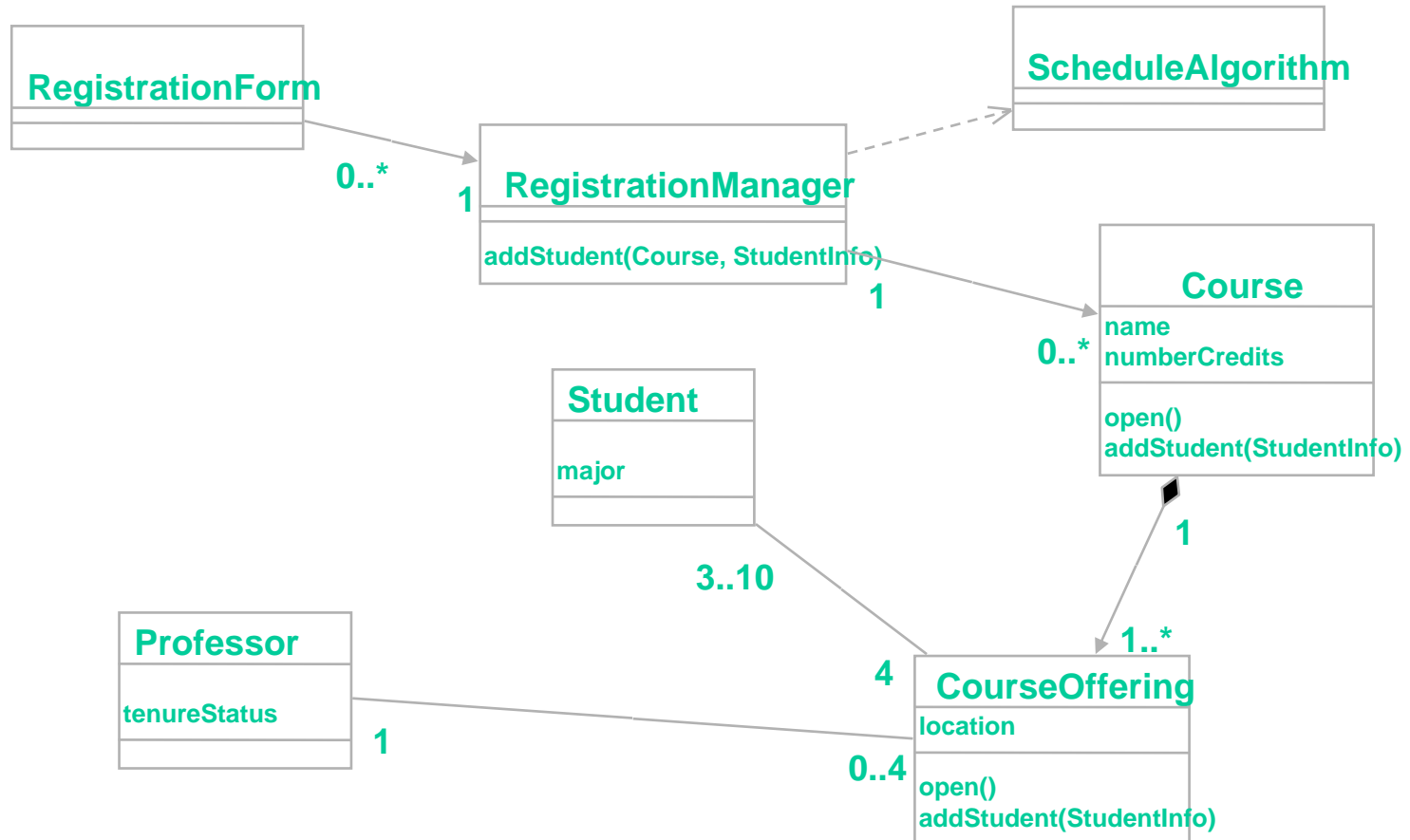
Course
name numberCredits
open() addStudent(StudentInfo)

Student
name major

Professor
name tenureStatus

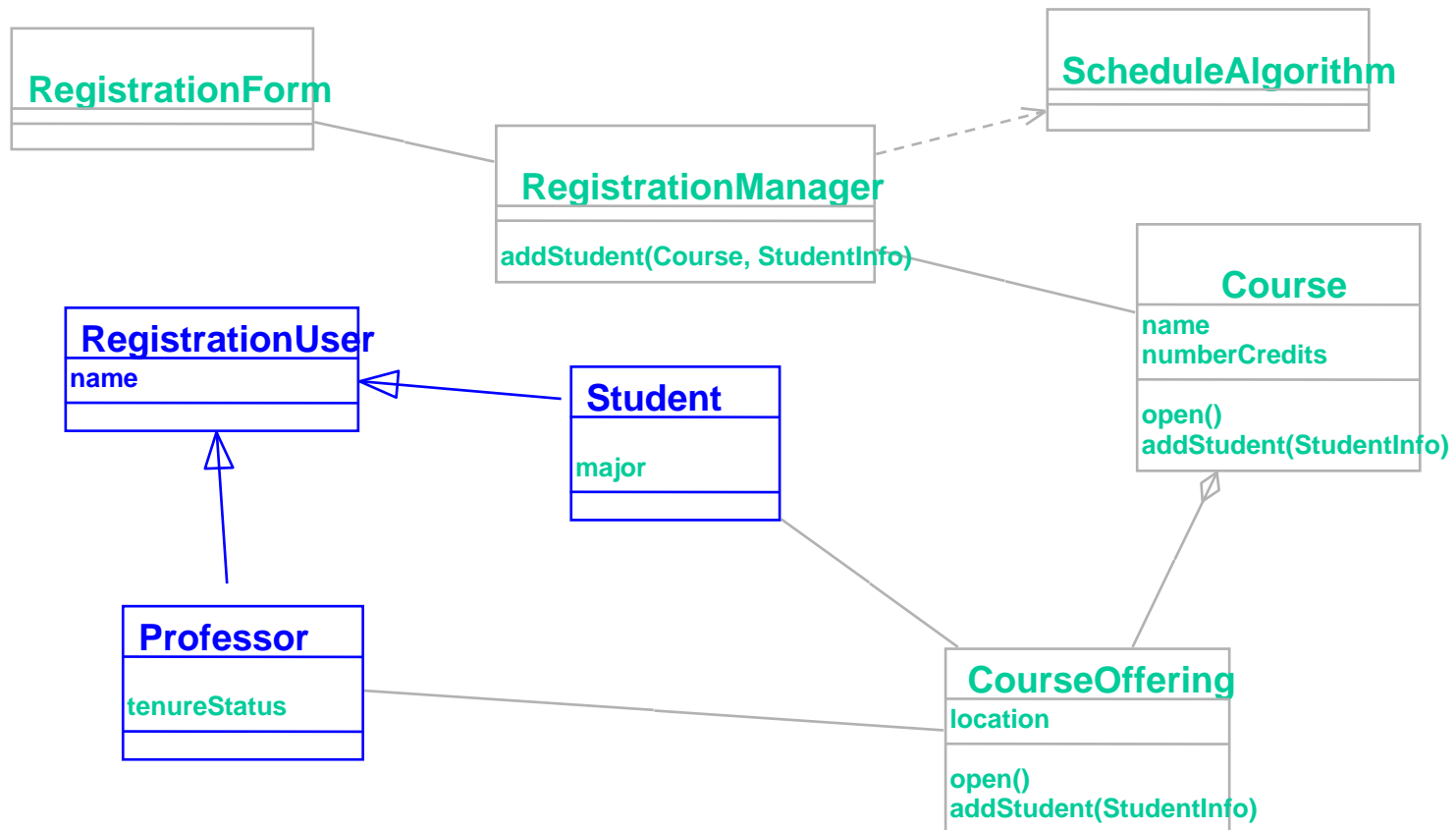
CourseOffering
location
open() addStudent(StudentInfo)

Class Relationships



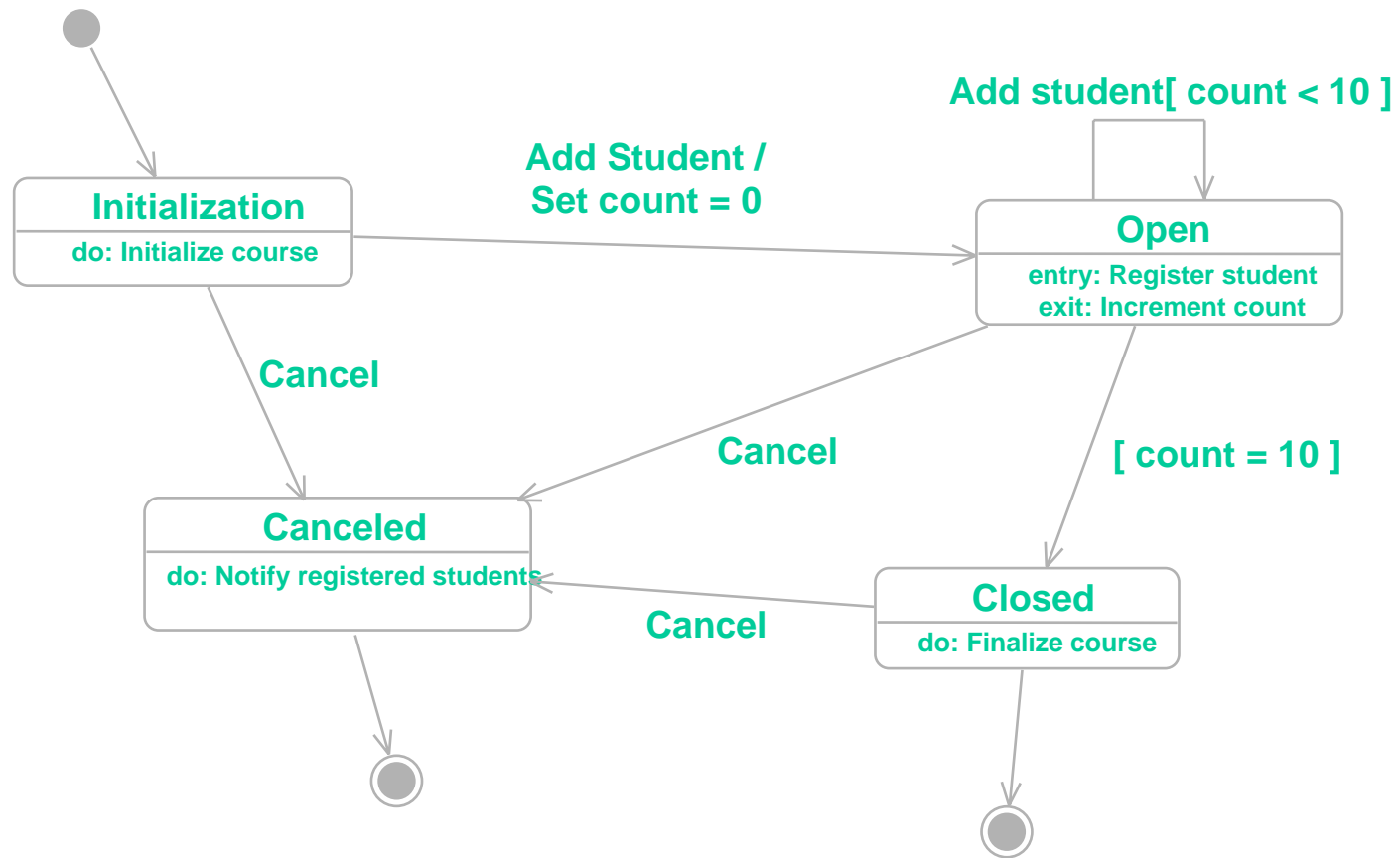
Class Inheritances

- Inheritance is a relationship between a superclass and its subclasses
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy



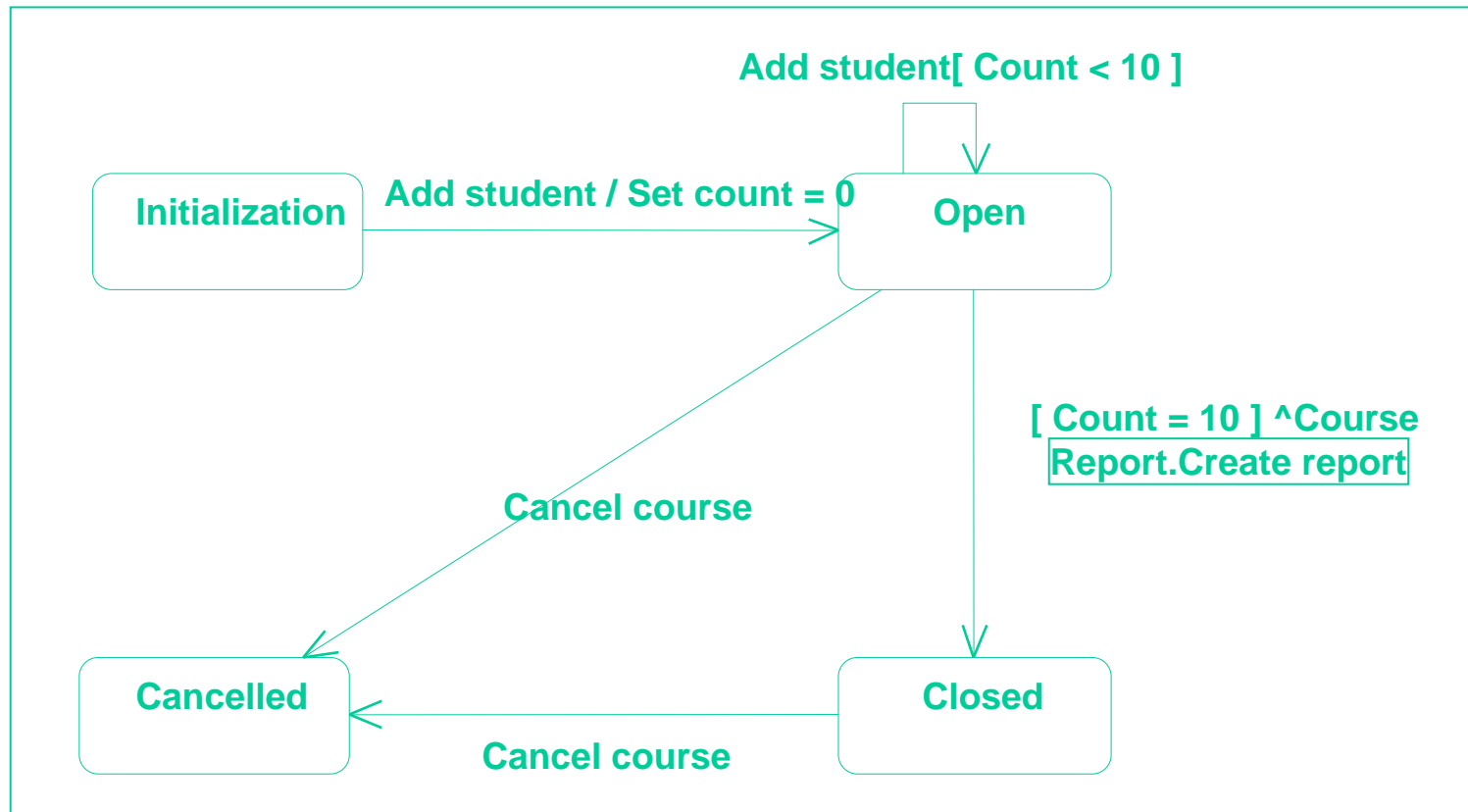
State Transition Diagram

- The life history of a given class
- The events that cause a transition from one state to another
- The actions that result from a state change



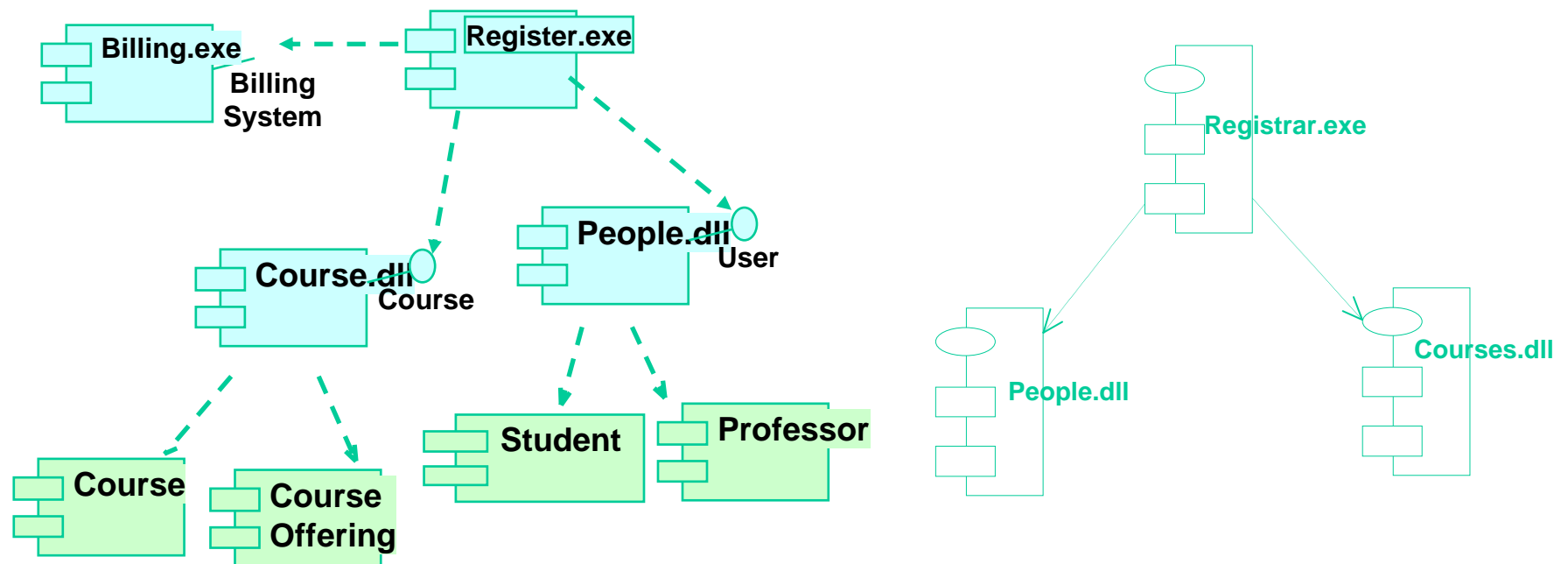
Statechart Diagram

- shows a state machine, consisting of states, transitions, events and activities



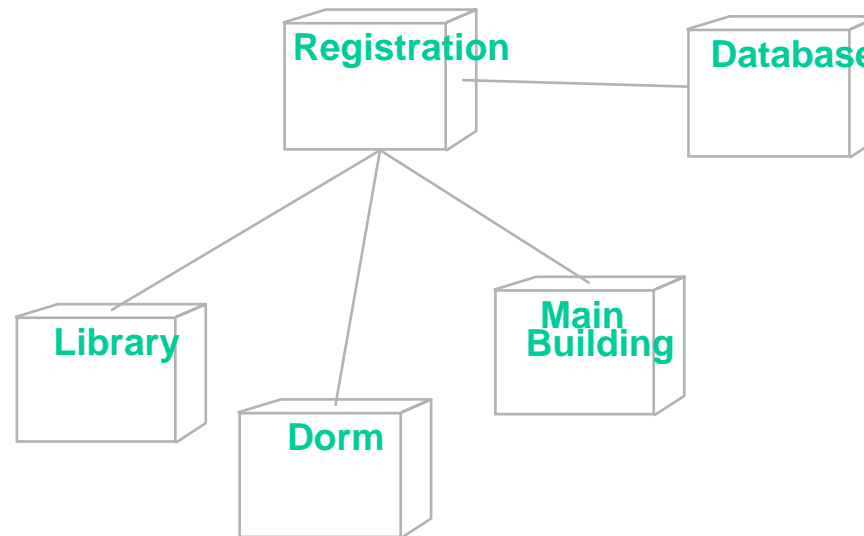
Component Diagram

- shows the organizations and dependencies among a set of components.



Deployment Diagram

- The deployment diagram shows the configuration of run-time processing elements and the software processes living on them.
- The deployment diagram visualizes the distribution of components across the enterprise.



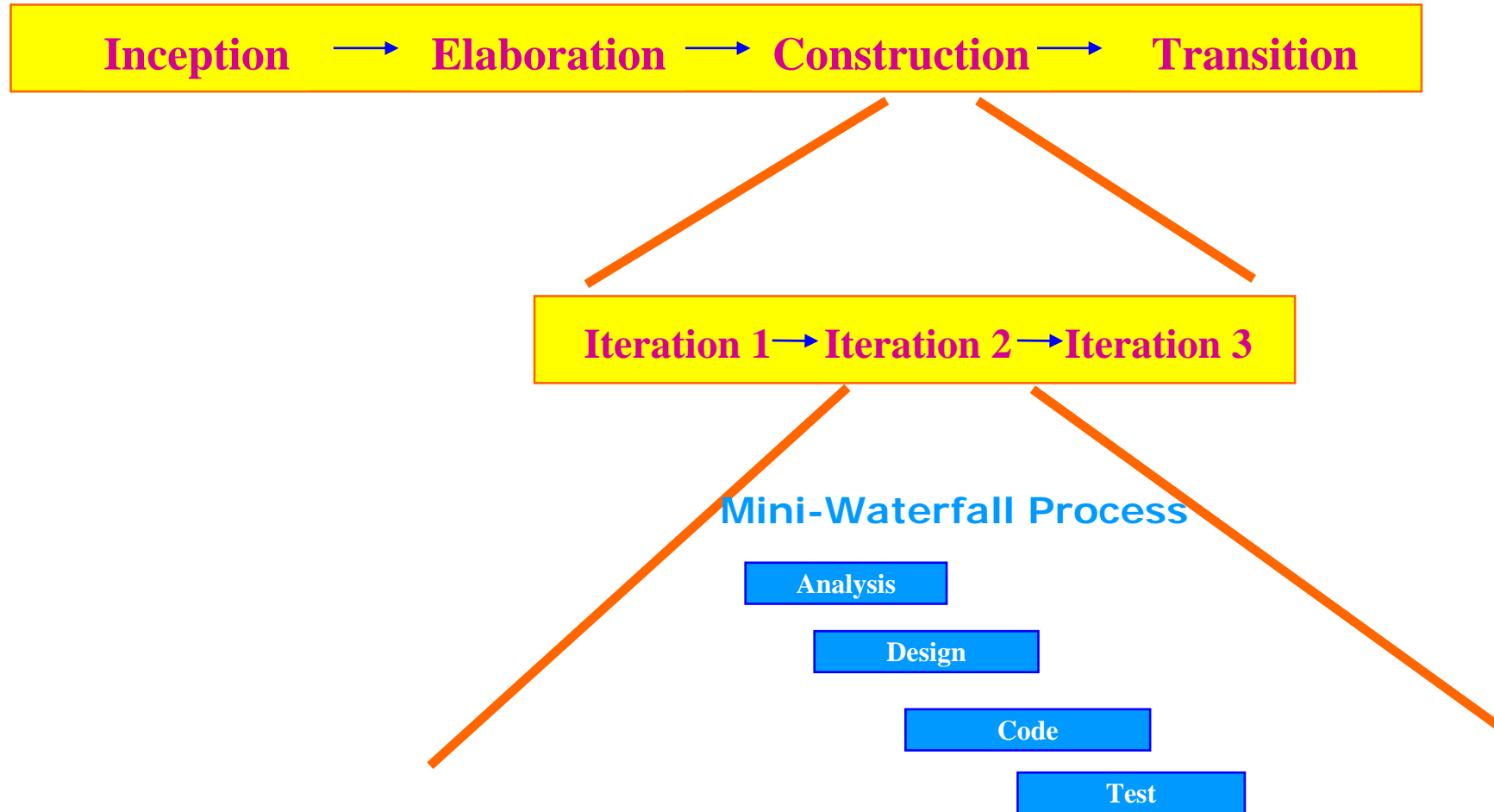
Guidelines to use UML

- Display the boundary of a system and its major functions using use cases and actors
- Illustrate use case realizations with interaction diagrams
- Represent a static structure of a system using class diagrams
- Model the behavior of objects with state transition diagrams
- Reveal the physical implementation architecture with component and deployment diagrams

UML Summary

- UML is effective for modeling large and complex software systems
- It is simple to learn for most developers, but provides advanced features for expert analysts and designers
- It can specify systems in an implementation-independent manner
- 10-20% of the constructs are used 80-90% of the time
- Structural modeling specifies a skeleton that can be refined and extended with additional structure and behavior
- Use case modeling specifies the functional requirements of system in an object-oriented manner

The Rational Unified Process (RUP)



Each iteration is defined in terms of the scenarios it implements

RUP Phases

- RUP is an iterative, incremental software development process model. (lifecycle)
- It can be used for object-oriented software development projects as complementary to the UML.
- **Inception:** seed idea is brought up to point of being a viable project. (%10)
- **Elaboration:** product vision and architecture are defined. (%30)
- **Construction:** brought from architectural baseline to point of deployment into user community. (%50)
- **Transition:** turned over to the user community.(%10)