

Basic Characteristics of Object-Oriented Systems

Basic Characteristics of Object Oriented Systems

- Classes and Objects
- Methods and Messages
- Encapsulation and Information Hiding
- Inheritance
- Polymorphism and Dynamic Binding

Classes and Objects

- Class – Template to define specific instances or objects
- Object – Objects of a class
- Attributes – Describes the object
- Behaviors – specify what object can do

Classes and Objects

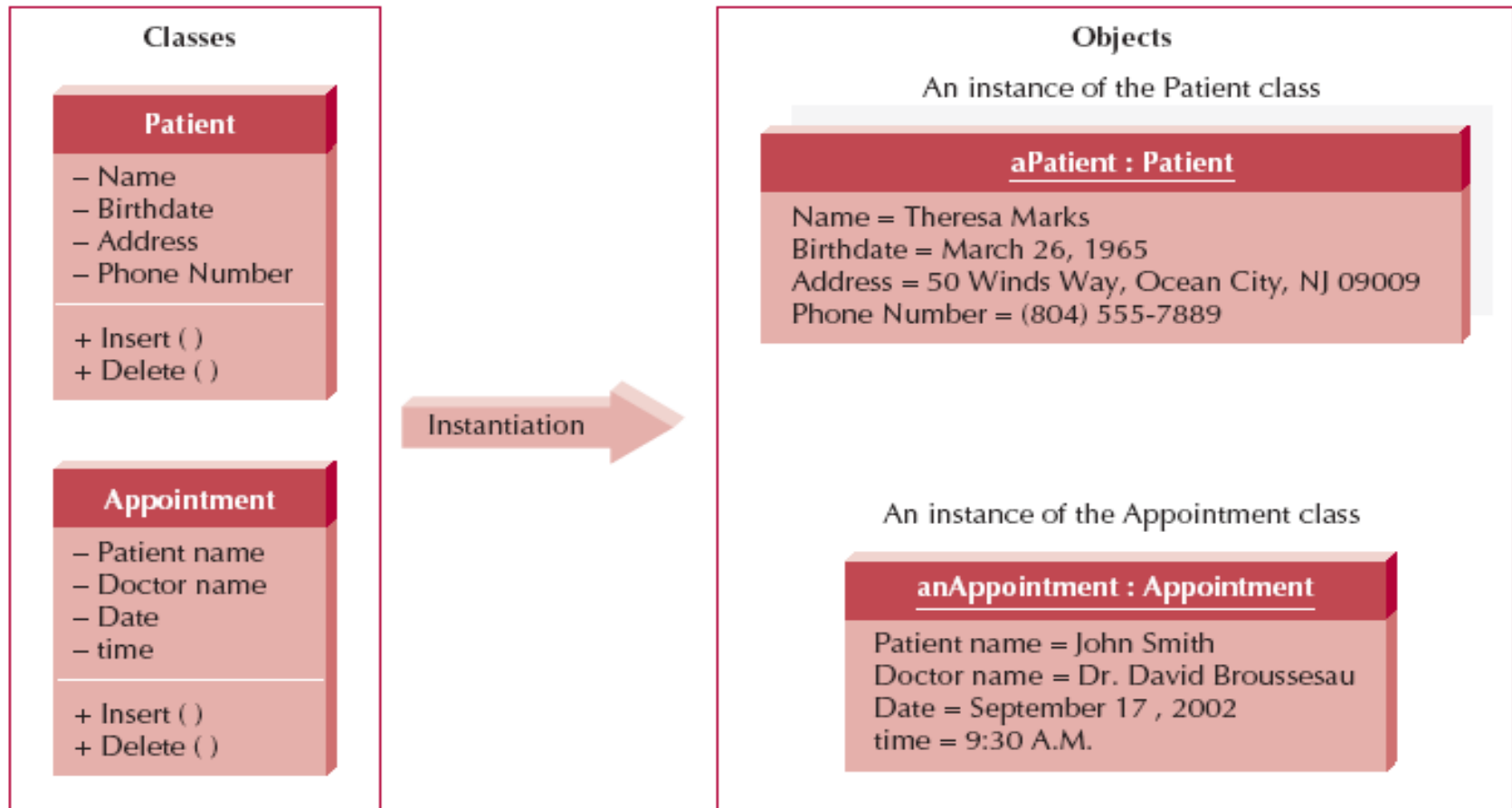
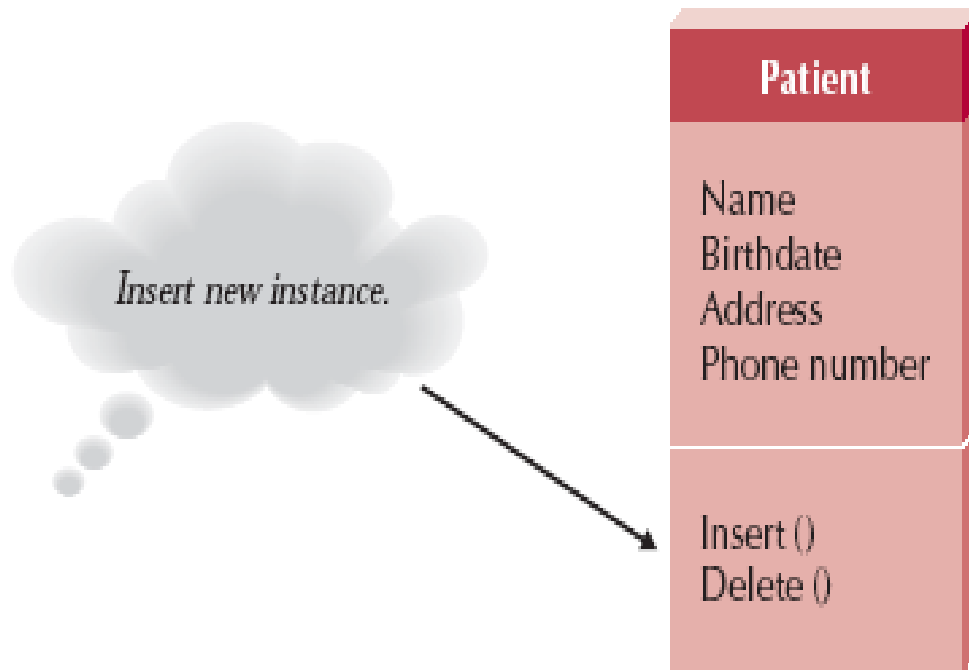


FIGURE 2-1 Classes and Objects

Methods and Messages

- Methods implement an object's behavior
 - Analogous to a function or procedure
- Messages are sent to trigger methods
 - Procedure call from one object to the next

Messages and Methods



A message is sent to the application.

The object's insert method will respond to the message and insert a new patient instance.

FIGURE 2-2

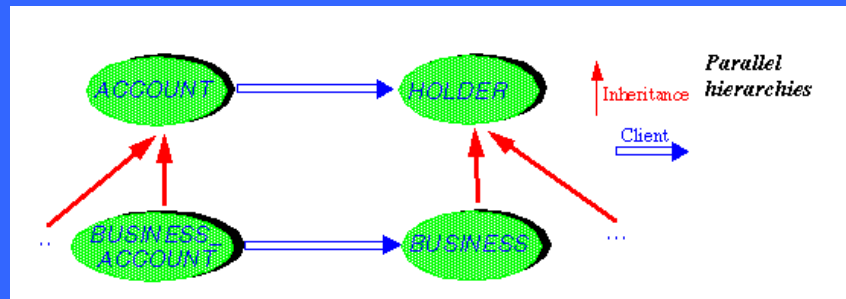
Messages and Methods

Encapsulation and Information Hiding

- Encapsulation
 - combination of data and process into an entity
- Information Hiding
 - Only the information required to use a software module is published to the user
- Reusability Key
 - Use an object by calling methods

Inheritance

- Superclasses or general classes are at the top of a hierarchy of classes
- Subclasses or specific classes are at the bottom
- Subclasses inherit attributes and methods from classes higher in the hierarchy



Class Hierarchy

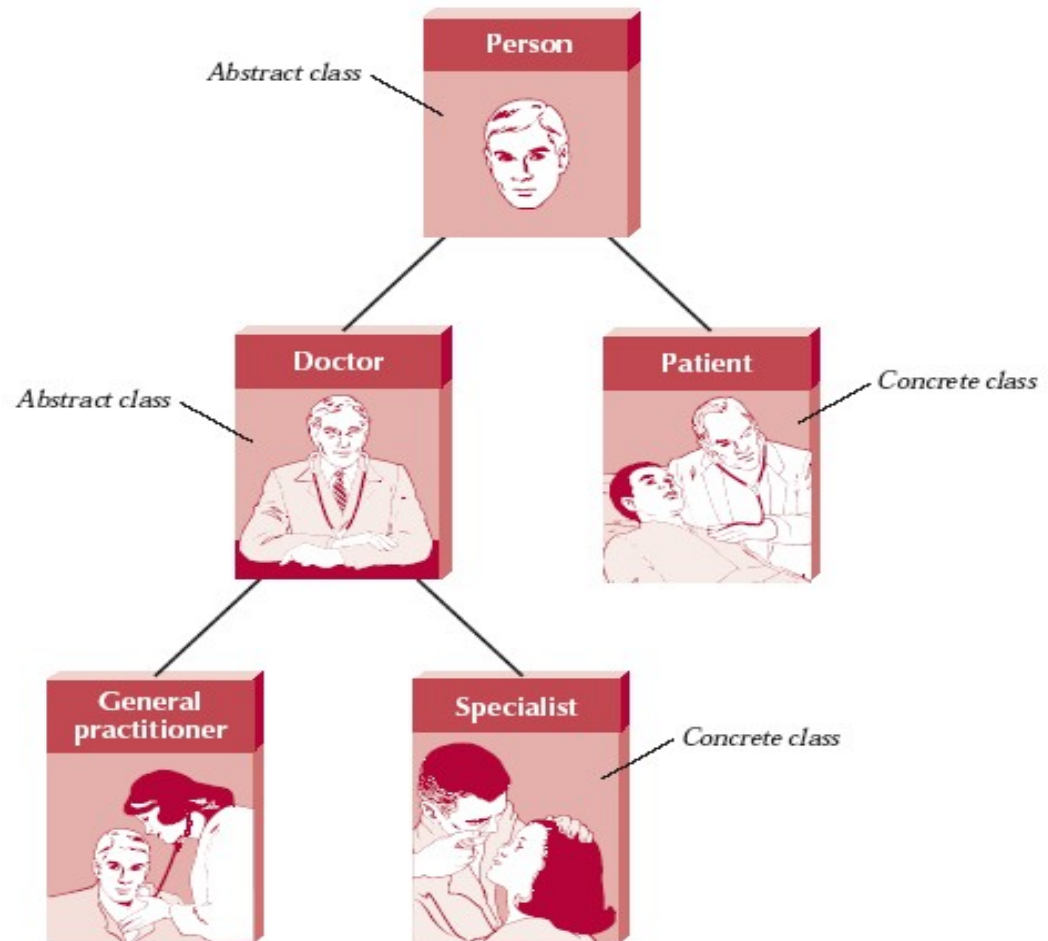


FIGURE 2-3
Class Hierarchy

Inheritance

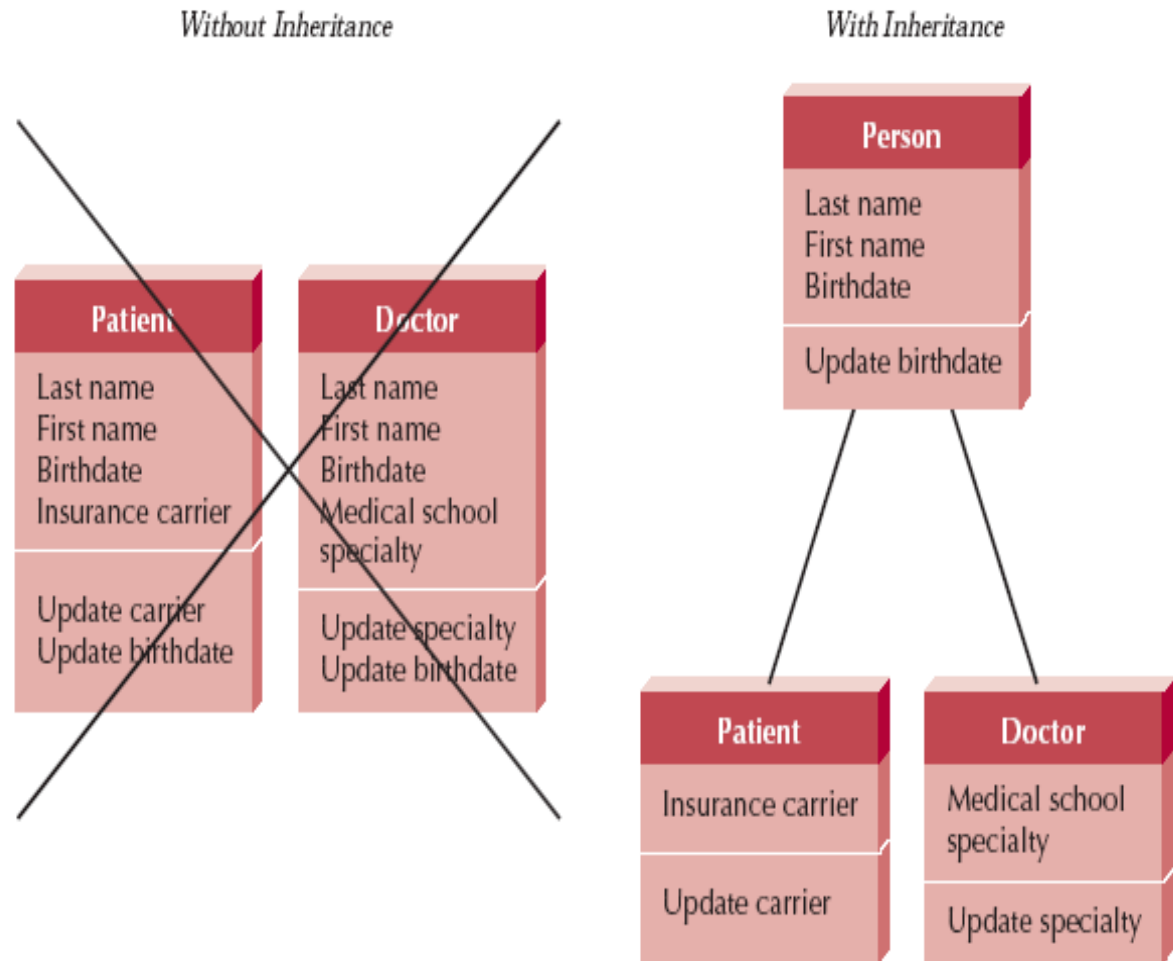
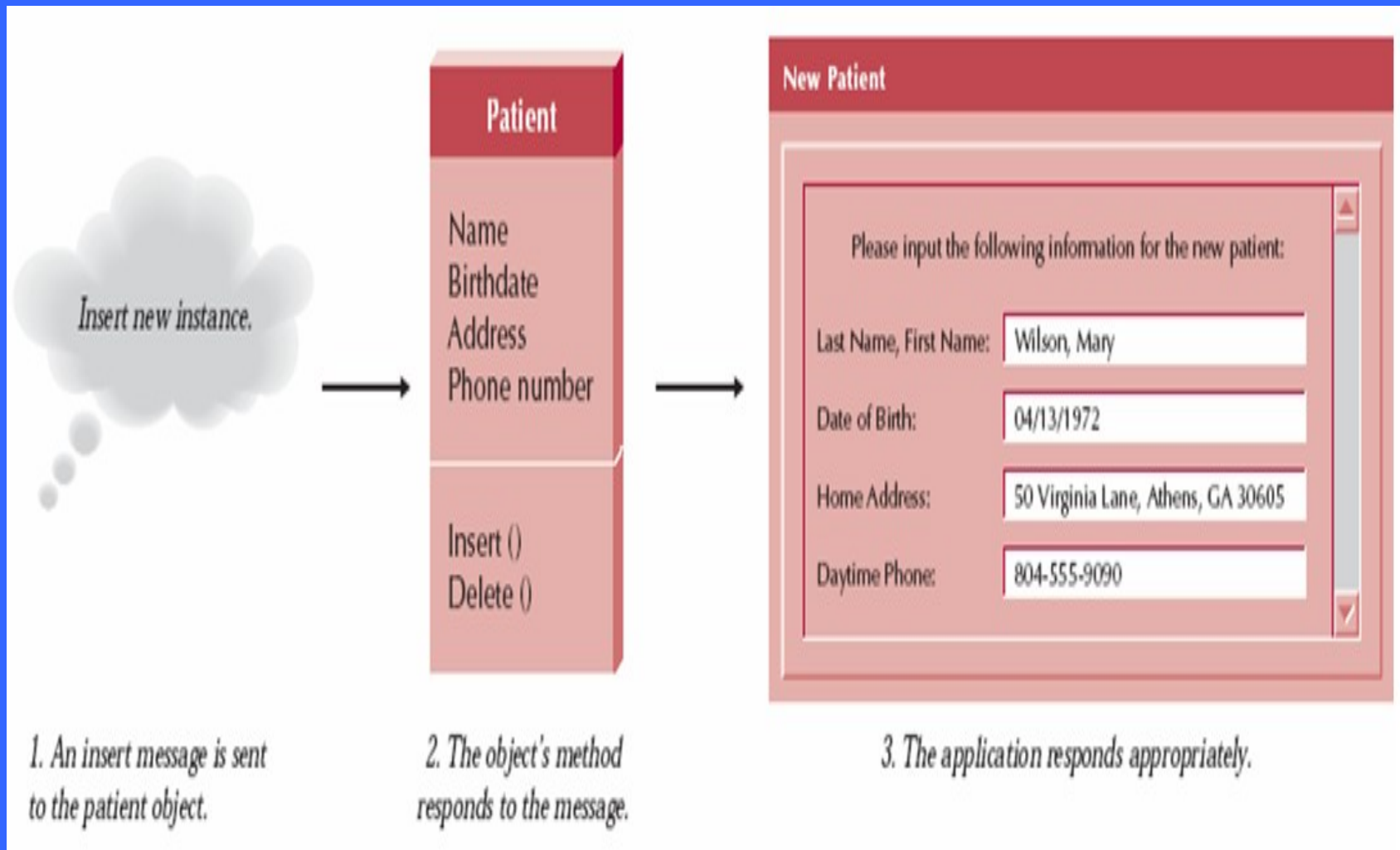


FIGURE 2-4
Inheritance

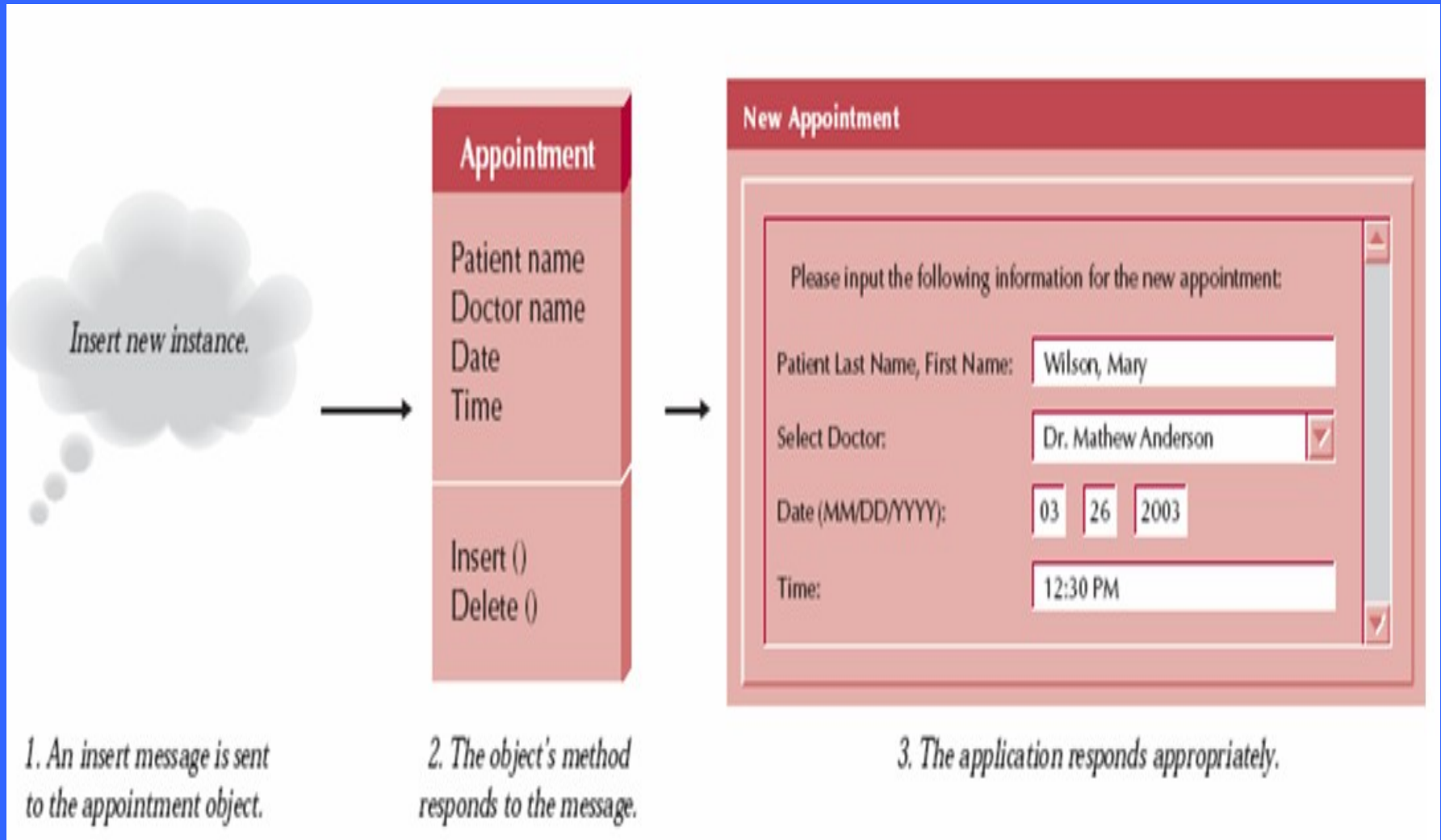
Polymorphism and Dynamic Binding

- Polymorphism
 - A message can be interpreted differently by different classes of objects
- Dynamic Binding
 - Sometimes called late binding
 - Delays typing or choosing a method for an object until run-time
- This is different from Static Binding
 - Type of object determined at compile time

Polymorphism and Dynamic Binding

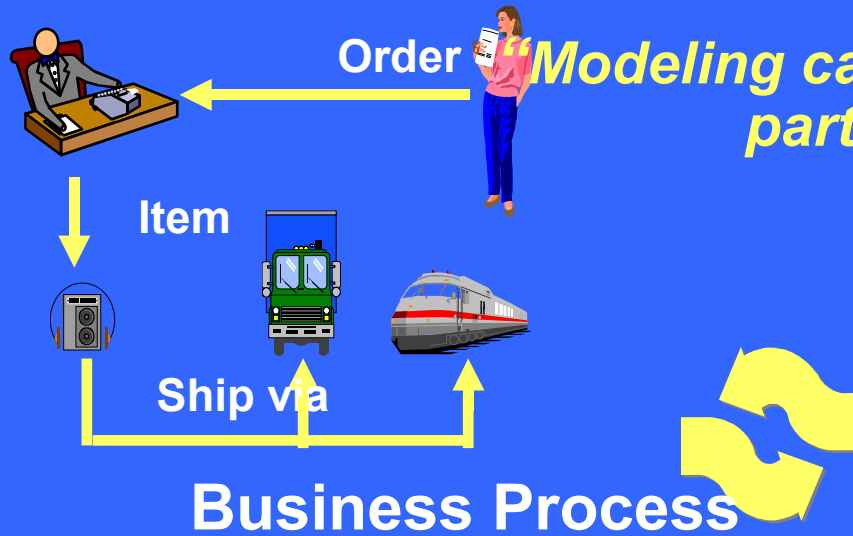


Polymorphism and Dynamic Binding

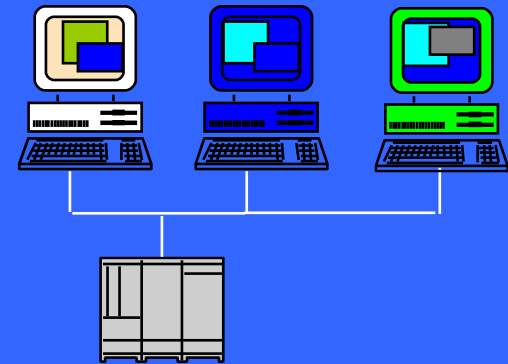


Analysis and Design with UML

What is Visual Modeling?



Modeling captures essential parts of the system.”
Dr. James Rumbaugh



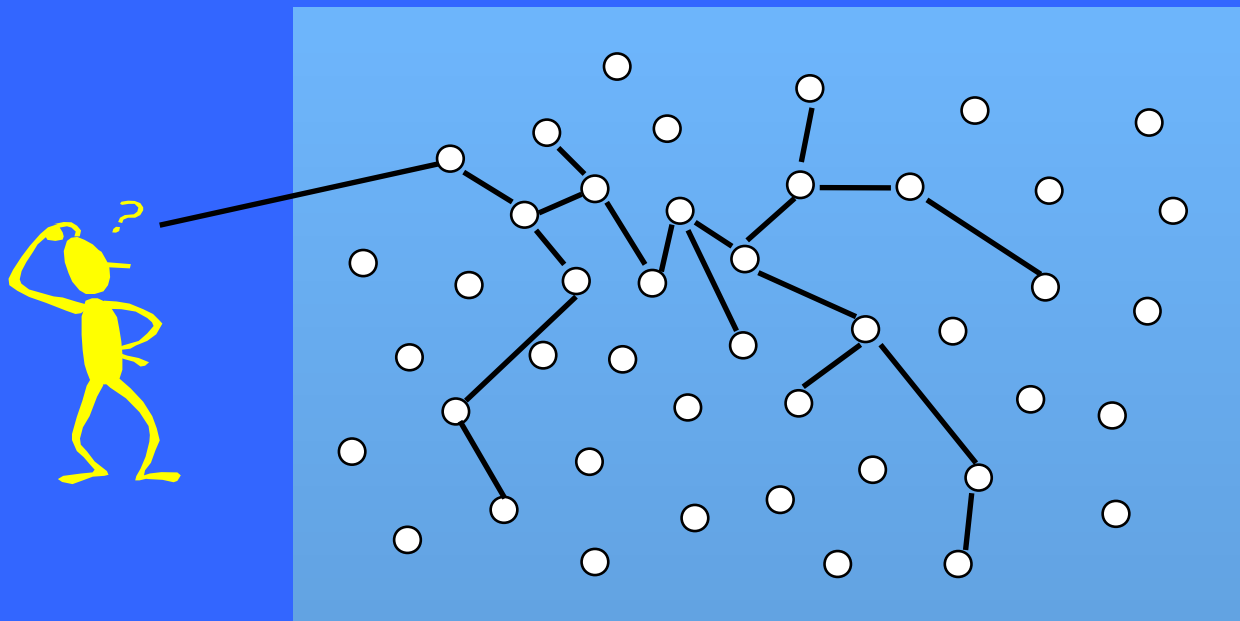
Computer System

Visual Modeling is modeling using standard graphical notations

Visual Modeling Captures

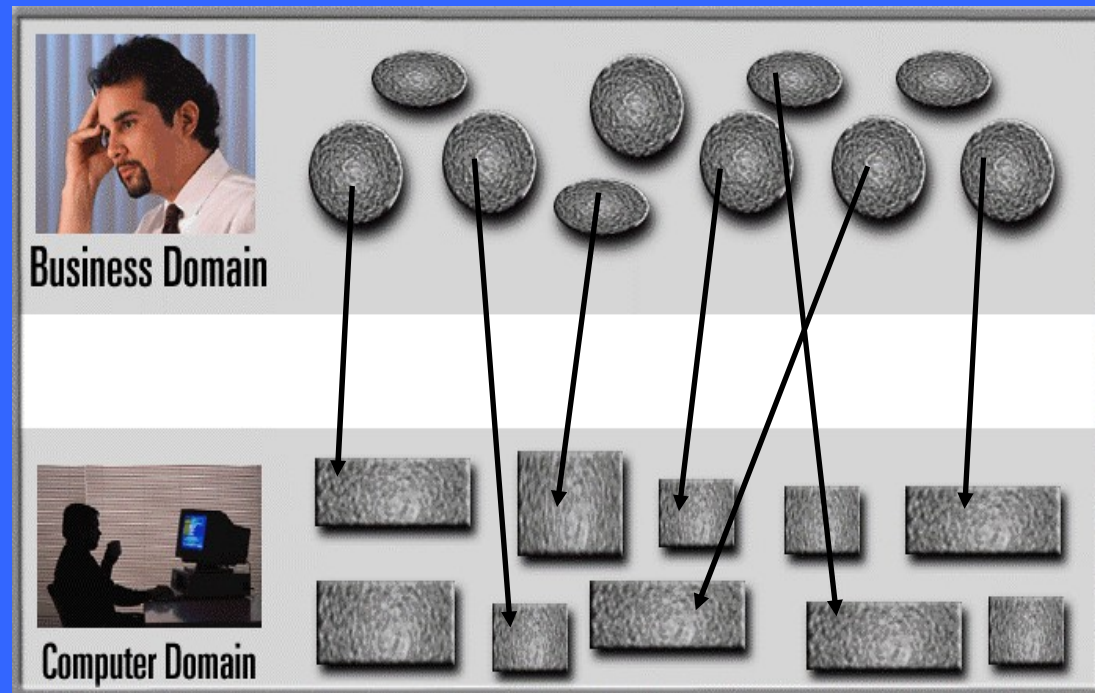
Use Case Analysis is a technique to capture business process from user's perspective

Business Process



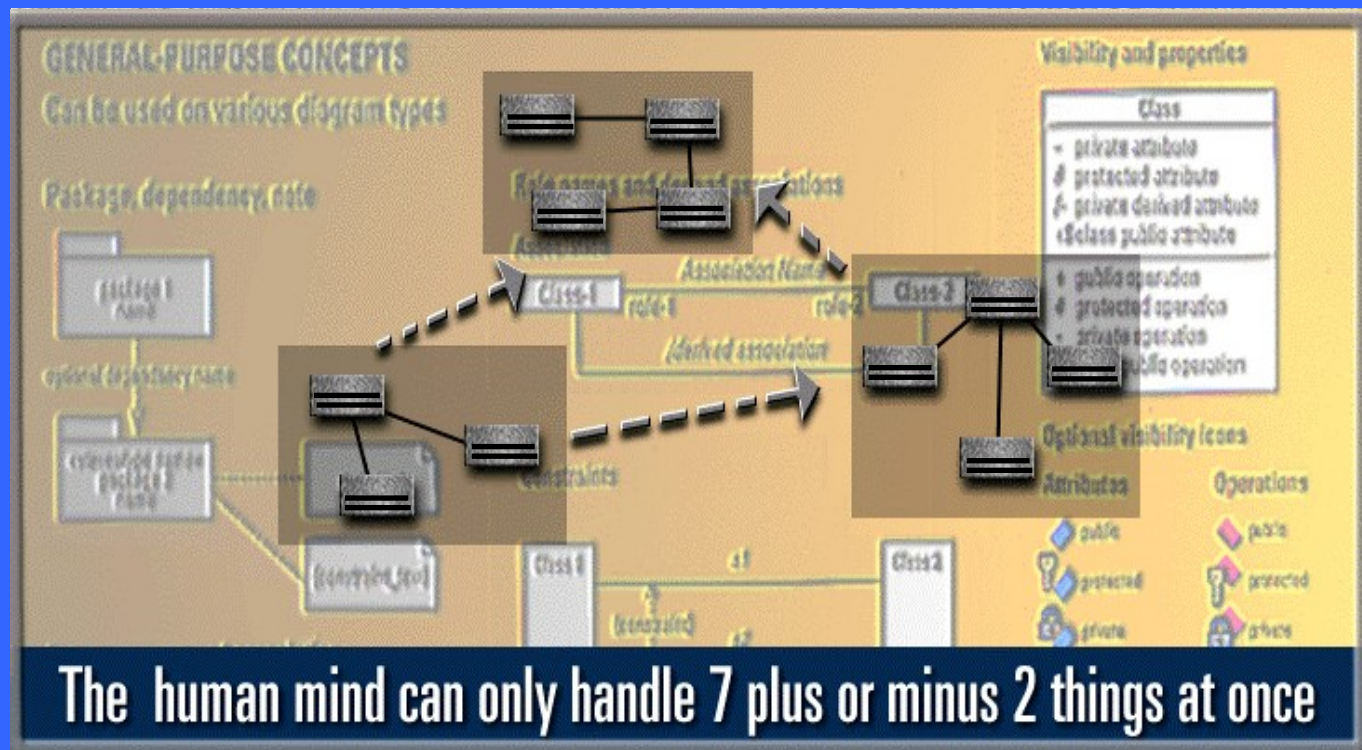
Visual Modeling is a Communication Tool

Use visual modeling to capture business objects and logic

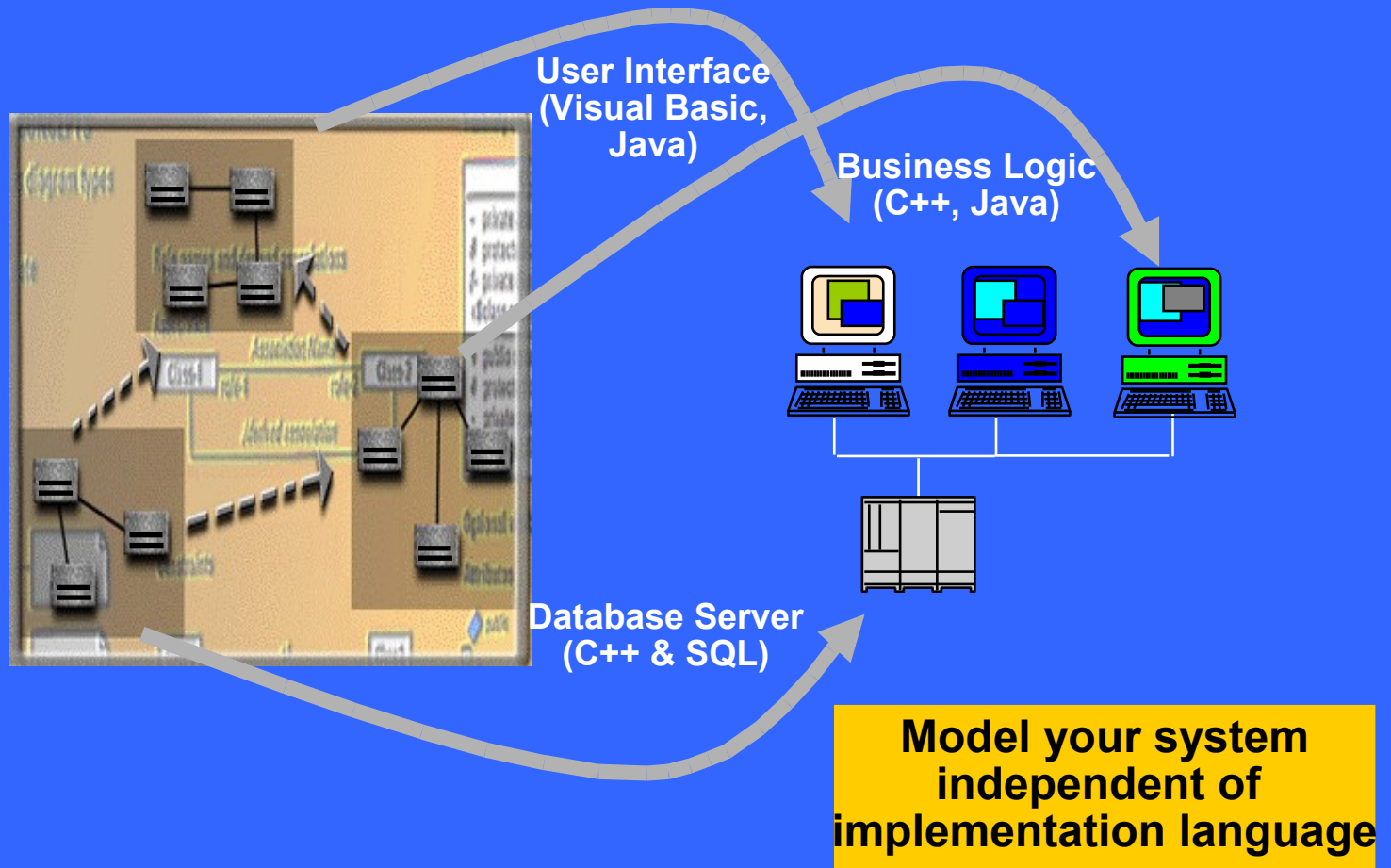


Use visual modeling to analyze and design your application

Visual Modeling Manages Complexity



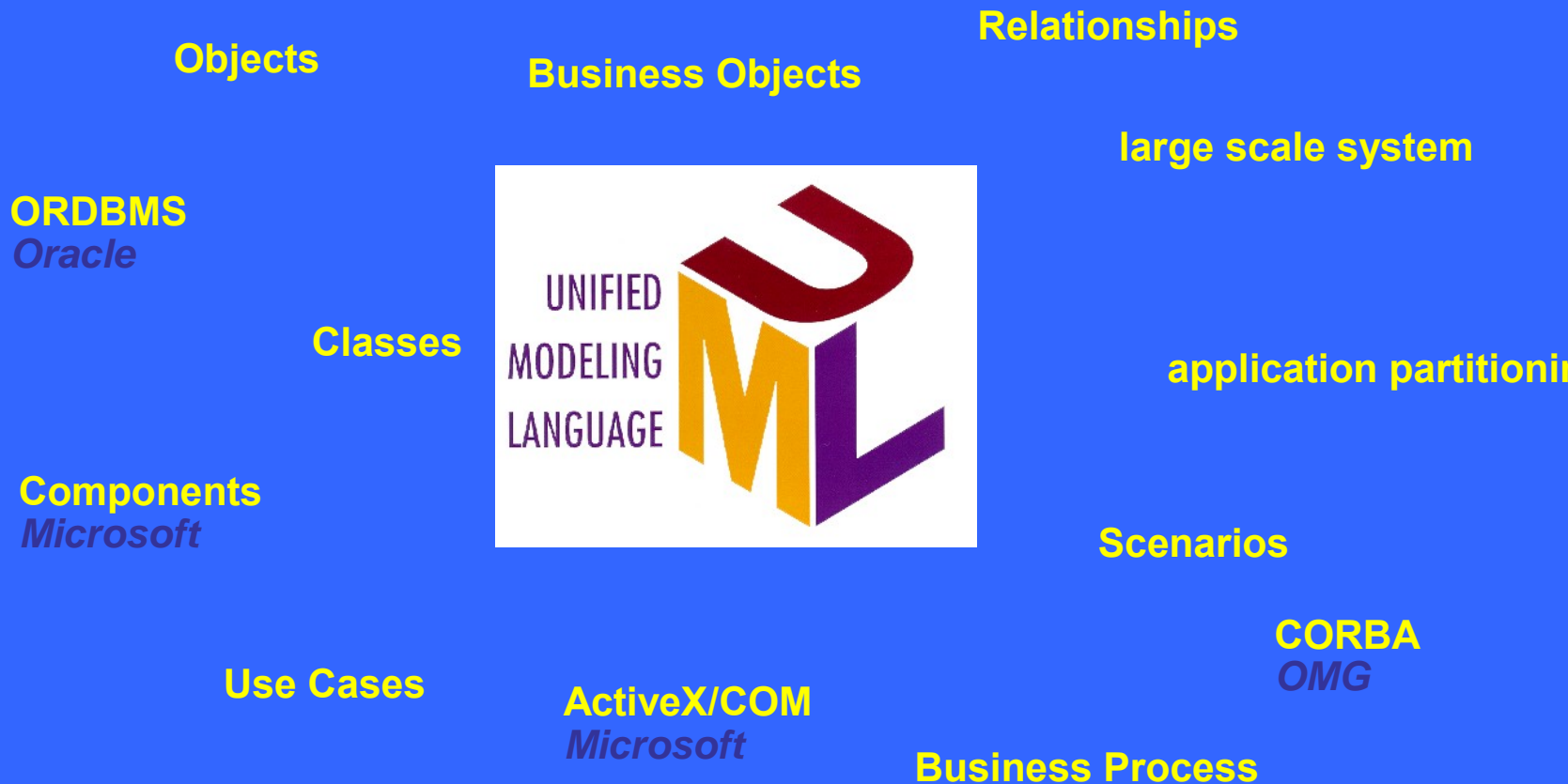
Visual Modeling Defines Software Architecture



What is the UML?

- UML stands for Unified Modeling Language
- The UML combines the best of the best from
 - Data Modeling concepts (Entity Relationship Diagrams)
 - Business Modeling (work flow)
 - Object Modeling
 - Component Modeling
- The UML is the standard language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system
- It can be used with all processes, throughout the development life cycle, and across different implementation technologies

UML Supports Application Development

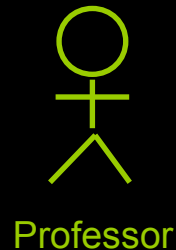
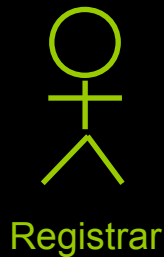


UML Concepts

- The UML may be used to:
 - Display the boundary of a system & 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 & deployment diagrams
 - Extend your functionality with stereotypes

Actors

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



Use Cases

- 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 -- request roster
 - Student -- maintain schedule
 - Billing System -- receive billing information from registration



Maintain Curriculum



Request Course Roster



Maintain Schedule

Documenting Use Cases

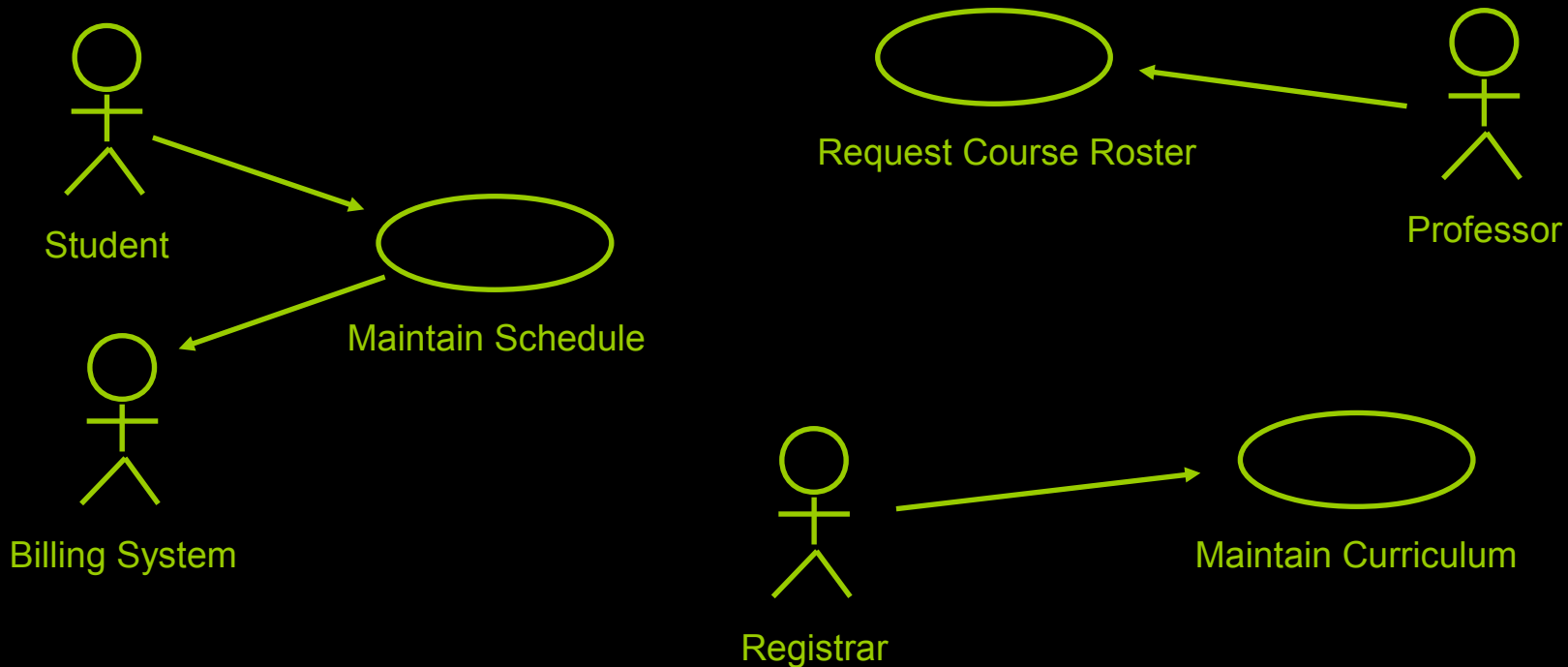
- A flow of events document is created for each use cases
 - Written from an actor point of view
- Details what the system must provide to the actor when the use cases is executed
- Typical contents
 - How the use case starts and ends
 - Normal flow of events
 - Alternate flow of events
 - Exceptional flow of events

Maintain Curriculum Flow of Events

- 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 (E-1) and prompts the Registrar to select the current semester or a future semester (E-2). The Registrar enters the desired semester. The system prompts the professor to select the desired activity: ADD, DELETE, REVIEW, or QUIT.
- If the activity selected is ADD, the S-1: Add a Course subflow is performed.
- If the activity selected is DELETE, the S-2: Delete a Course subflow is performed.
- If the activity selected is REVIEW, the S-3: Review Curriculum subflow is performed.
- If the activity selected is QUIT, the use case ends.
- ...

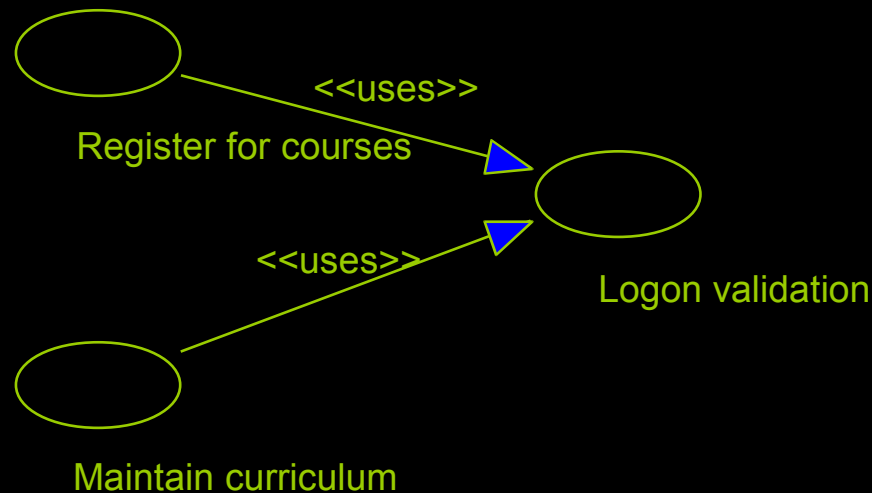
Use Case Diagram

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



Uses and Extends Use Case Relationships

- As the use cases are documented, other use case relationships may be discovered
 - A uses relationship shows behavior that is common to one or more use cases
 - An extends relationship shows optional behavior

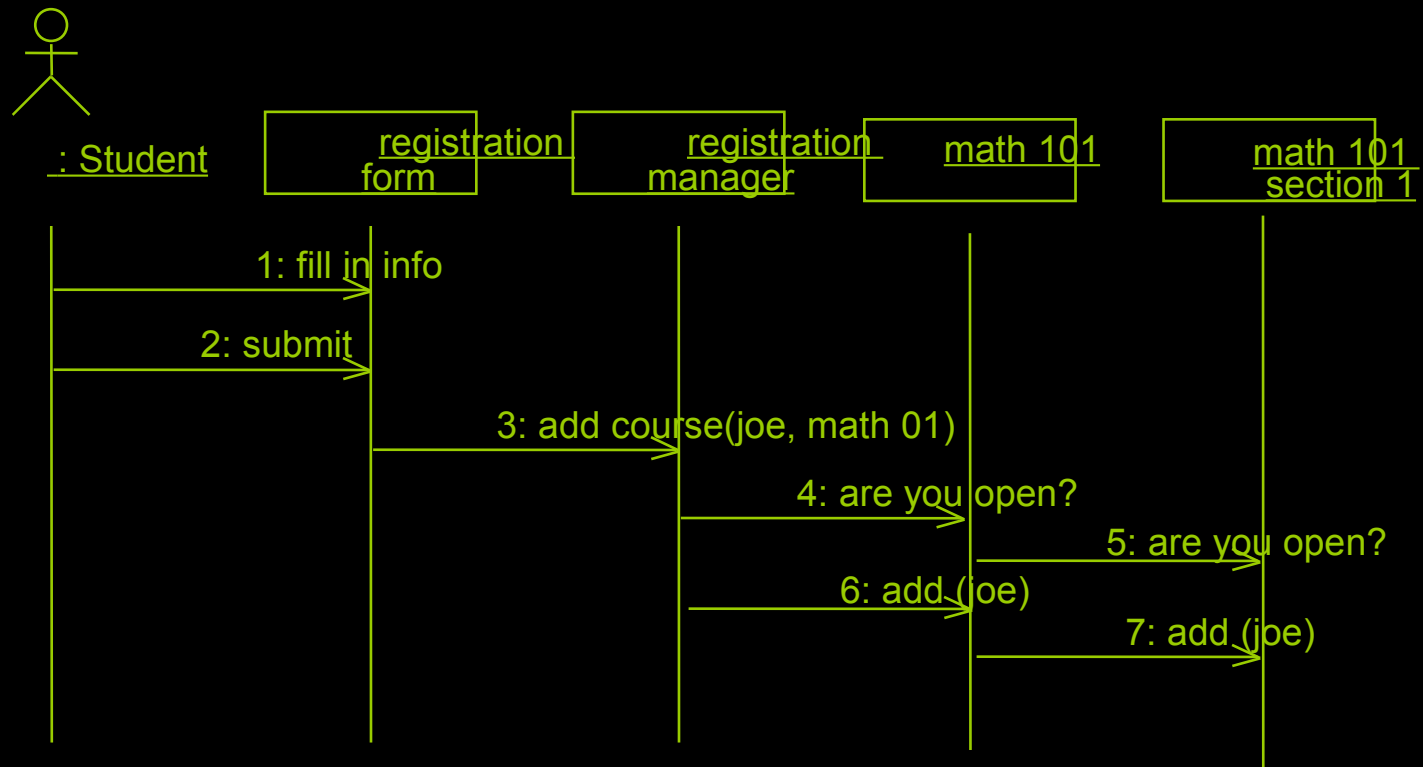


Use Case Realizations

- The use case diagram presents an outside view of the system
- Interaction diagrams describe how use cases are realized as interactions among societies of objects
- Two types of interaction diagrams
 - Sequence diagrams
 - Collaboration diagrams

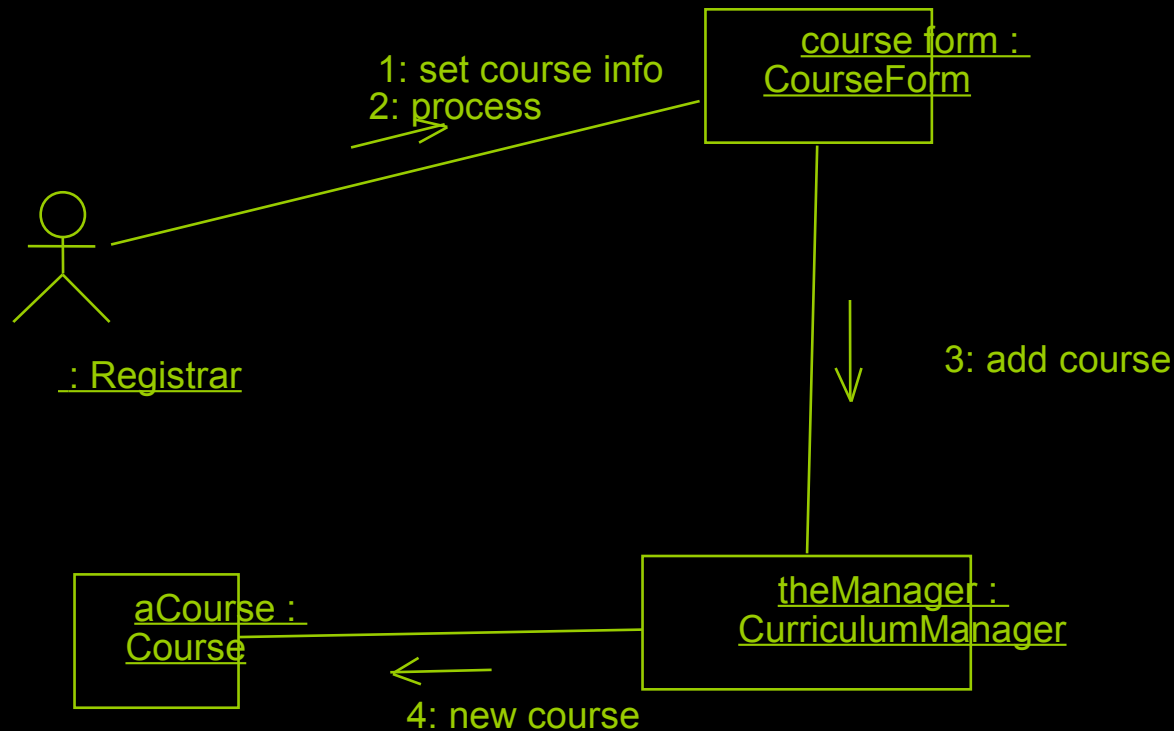
Sequence Diagram

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



Collaboration Diagram

- A collaboration diagram displays object interactions organized around objects and their links to one another



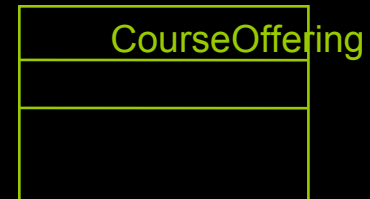
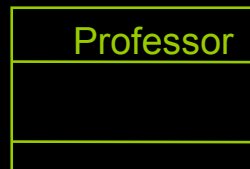
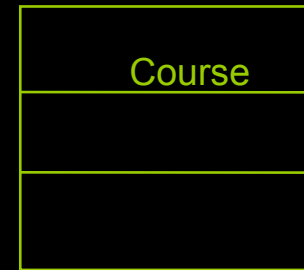
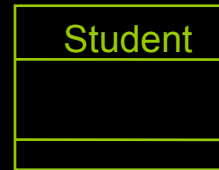
Class Diagrams

- 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

Classes

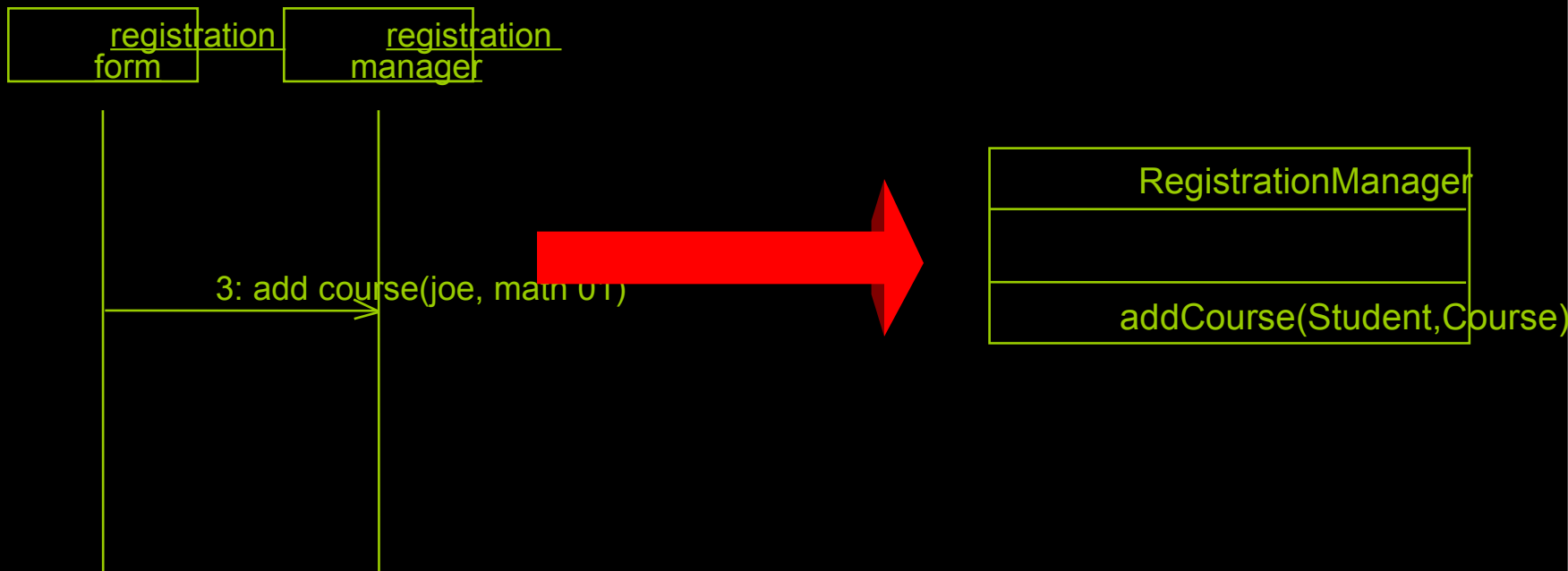
- A class is a collection of objects with common structure, common behavior, common relationships and common semantics
- Classes are found by examining the objects in sequence and collaboration diagram
- A class is drawn as a rectangle with three compartments
- Classes should be named using the vocabulary of the domain
 - Naming standards should be created
 - e.g., all classes are singular nouns starting with a capital letter

Classes



Operations

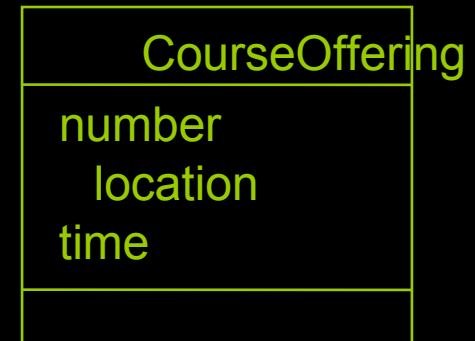
- The behavior of a class is represented by its operations
- Operations may be found by examining interaction diagrams



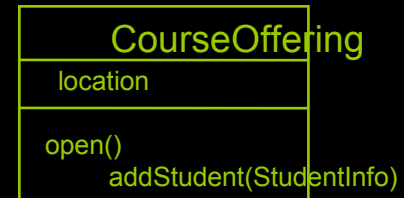
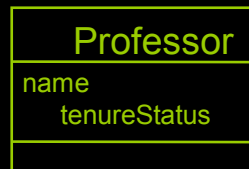
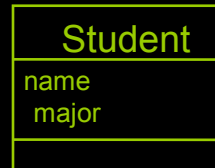
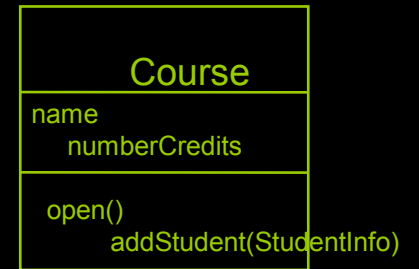
Attributes

- 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

Each course offering
has a number,
and time



Classes



Relationships

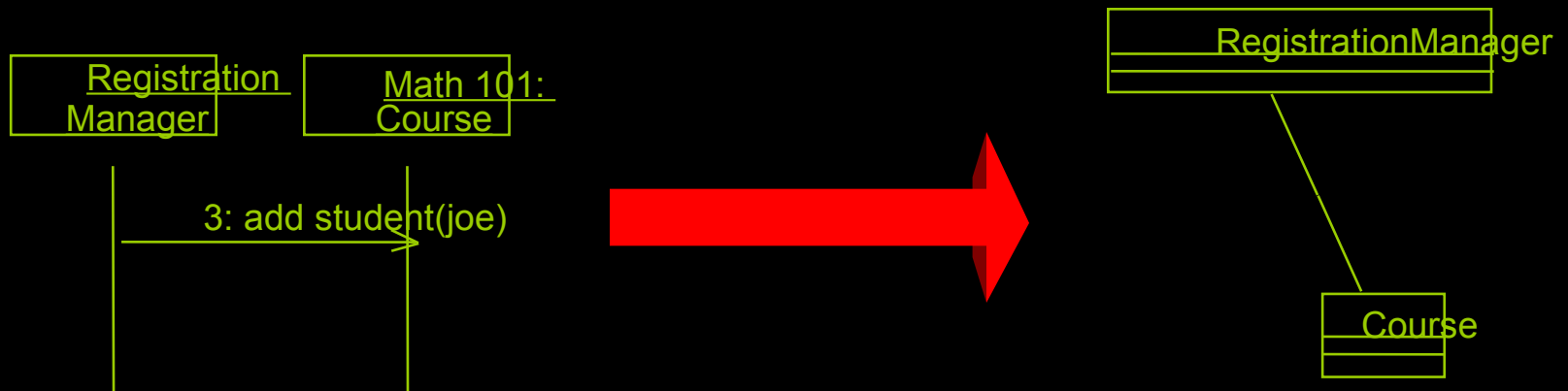
- Relationships provide a pathway for communication between objects
- Sequence and/or collaboration diagrams are examined to determine what links between objects need to exist to accomplish the behavior -- if two objects need to “talk” there must be a link between them
- Three types of relationships are:
 - Association
 - Aggregation
 - Dependency

Relationships

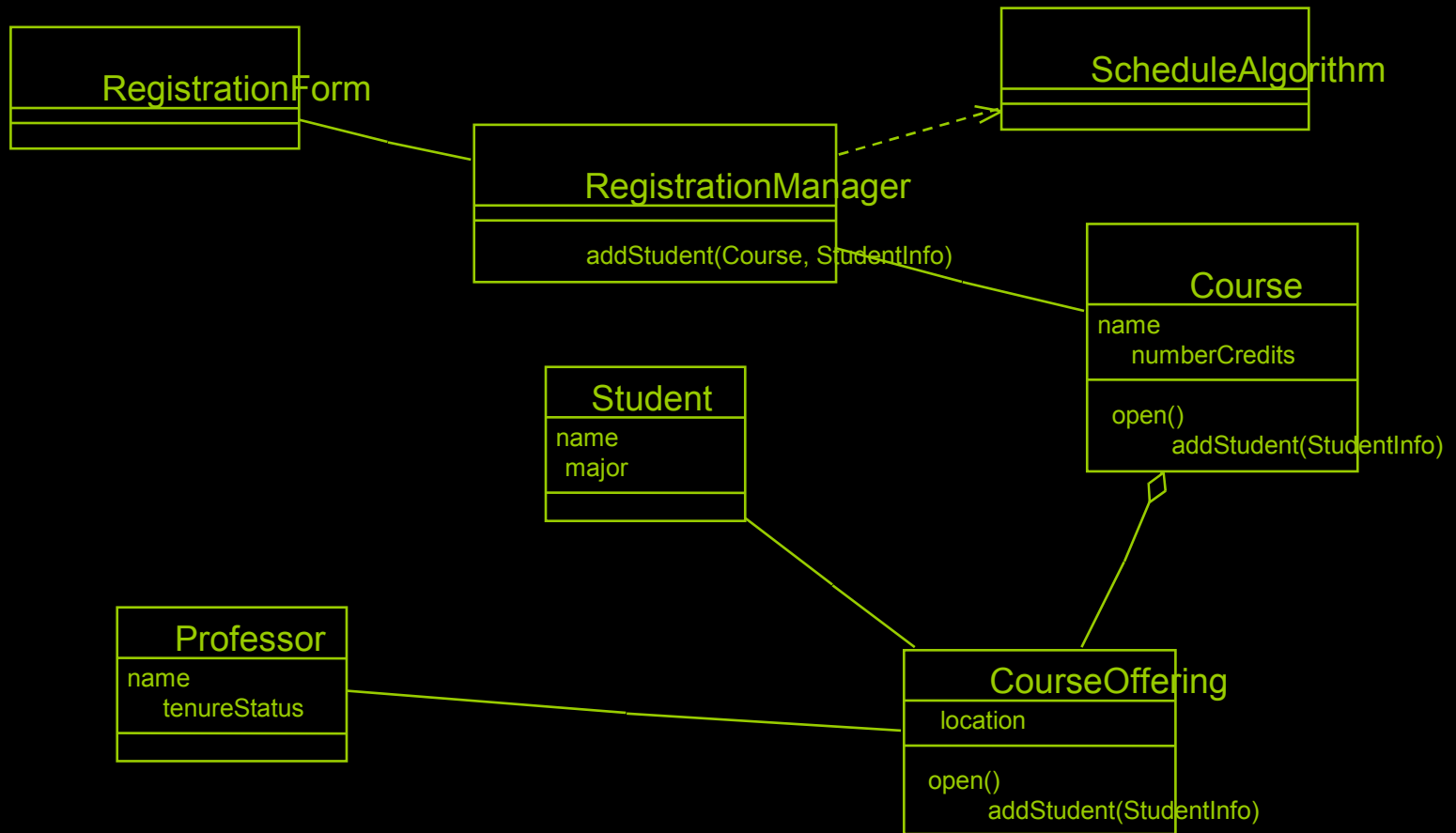
- An association is a bi-directional connection between classes
 - An association is shown as a line connecting the related classes
- An aggregation is a stronger form of relationship where the relationship is between a whole and its parts
 - An aggregation is shown as a line connecting the related classes with a diamond next to the class representing the whole
- A dependency relationship is a weaker form of relationship showing a relationship between a client and a supplier where the client does not have semantic knowledge of the supplier
- A dependency is shown as a dashed line pointing from the client to the supplier

Finding Relationships

- Relationships are discovered by examining interaction diagrams
 - If two objects must “talk” there must be a pathway for communication



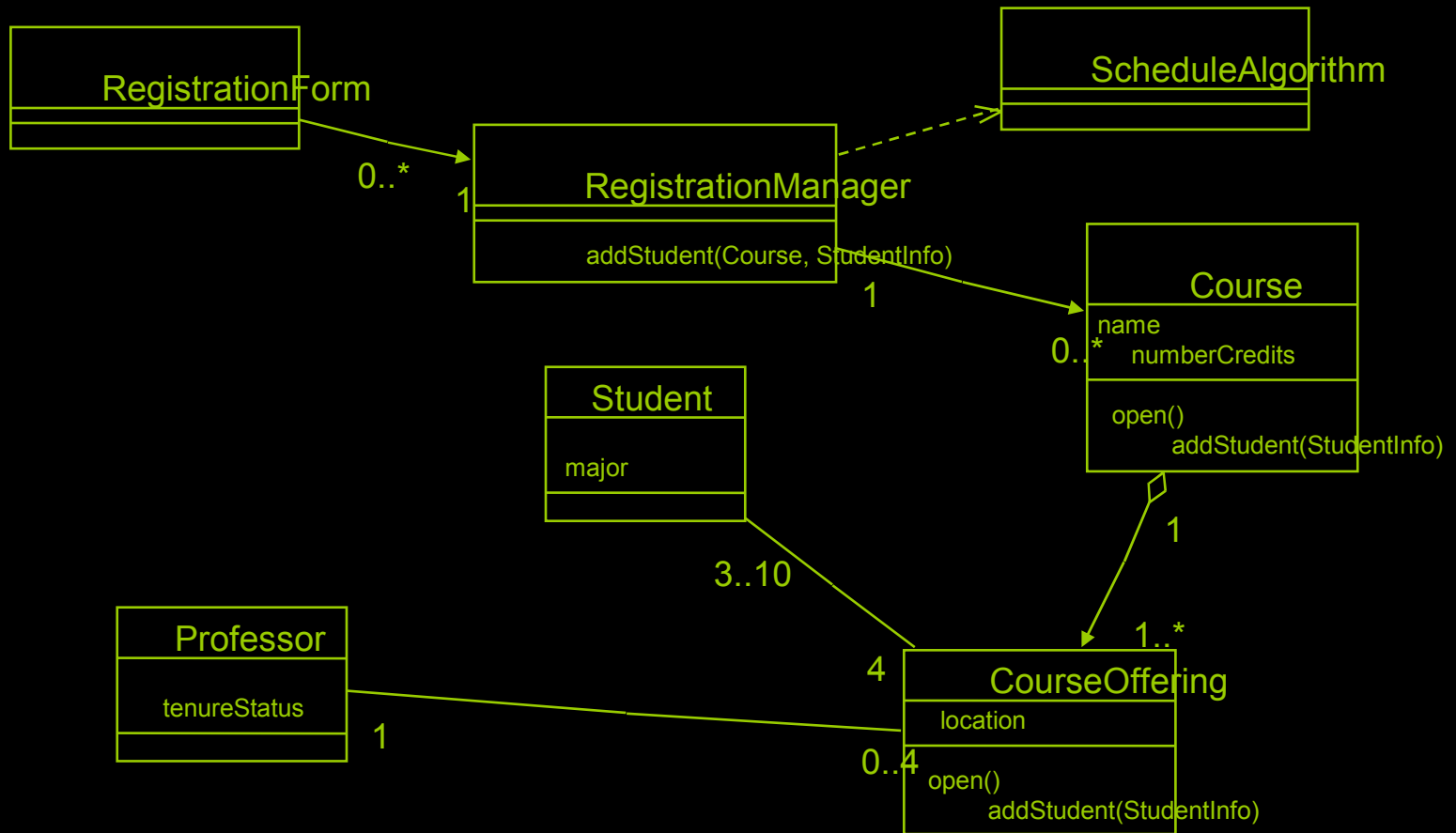
Relationships



Multiplicity and Navigation

- Multiplicity defines how many objects participate in a relationships
 - Multiplicity is the number of instances of one class related to ONE instance of the other class
 - For each association and aggregation, there are two multiplicity decisions to make: one for each end of the relationship
- Although associations and aggregations are bi-directional by default, it is often desirable to restrict navigation to one direction
- If navigation is restricted, an arrowhead is added to indicate the direction of the navigation

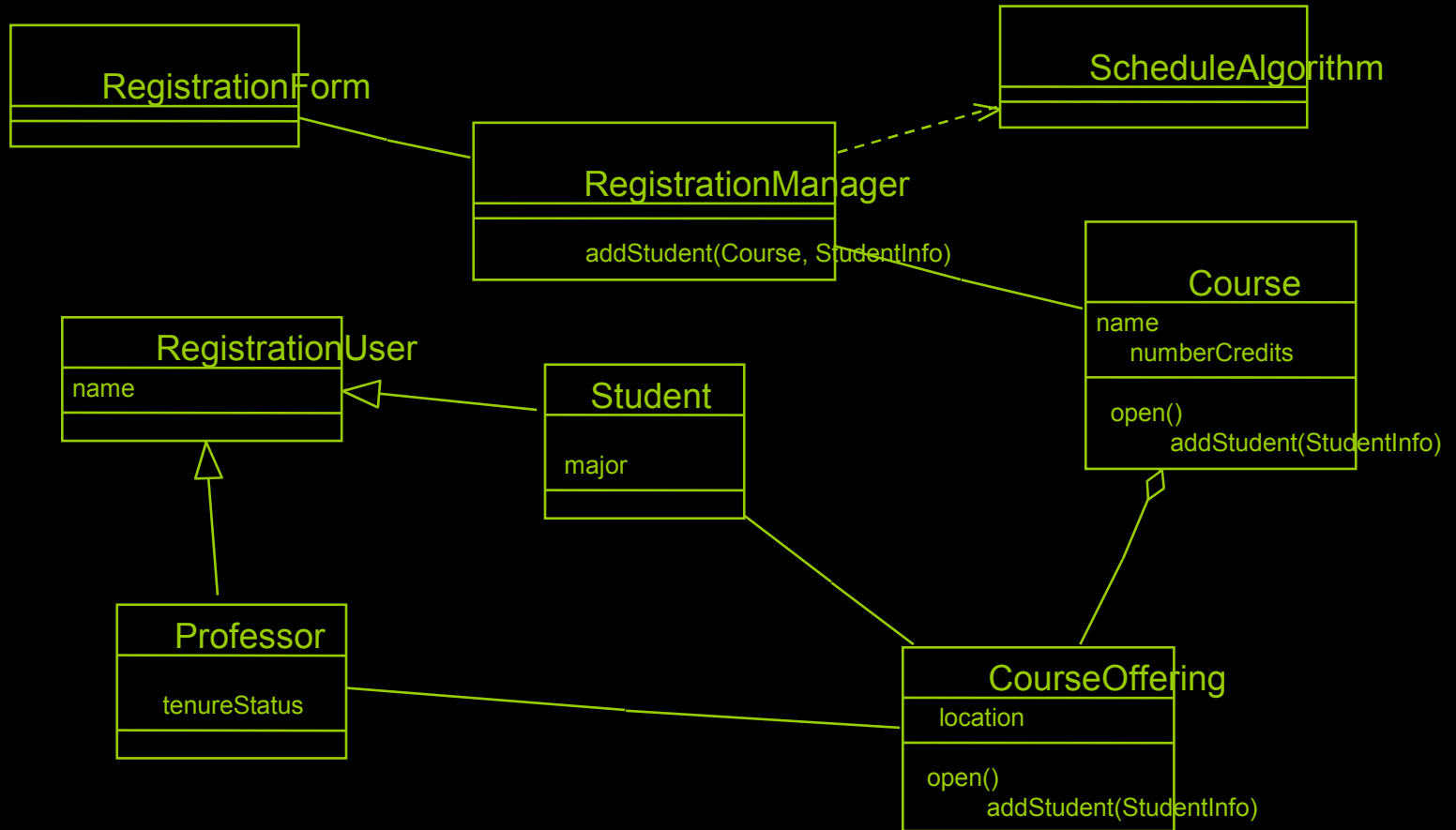
Multiplicity and Navigation



Inheritance

- Inheritance is a relationships between a superclass and its subclasses
- There are two ways to find inheritance:
 - Generalization
 - Specialization
- Common attributes, operations, and/or relationships are shown at the highest applicable level in the hierarchy

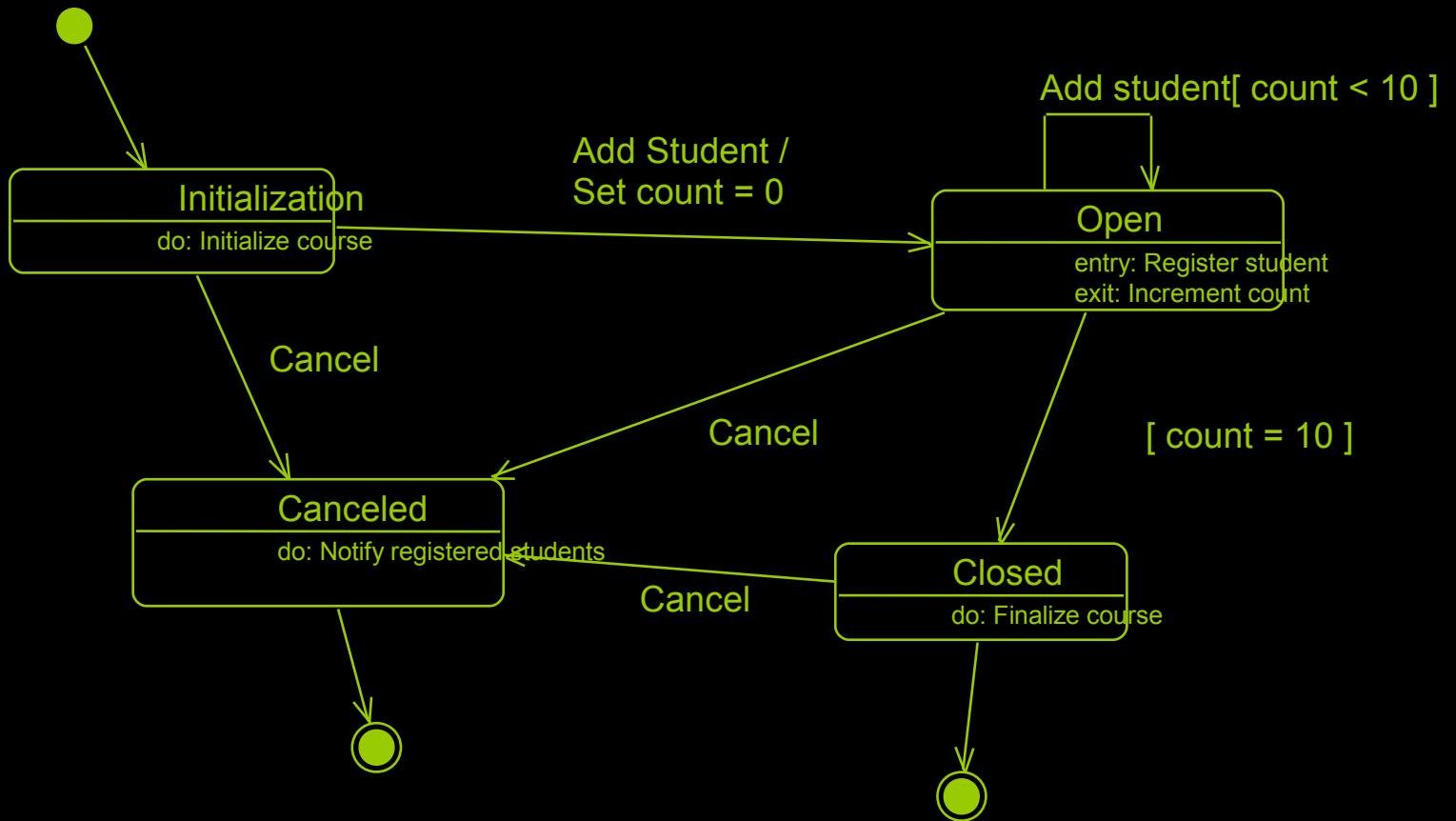
Inheritance



The State of an Object

- A state transition diagram shows
 - 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
- State transition diagrams are created for objects with significant dynamic behavior

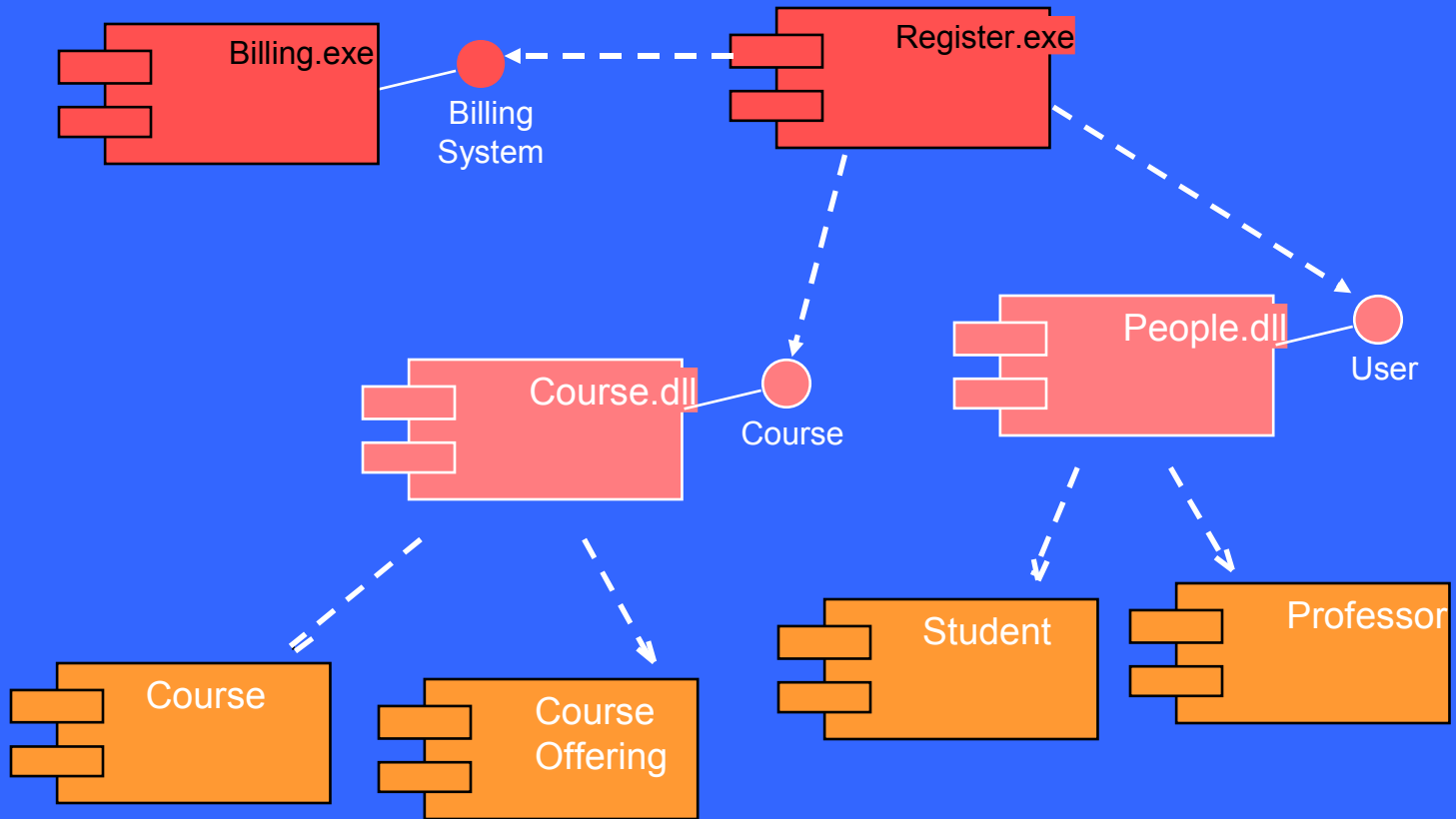
State Transition Diagram



The Physical World

- Component diagrams illustrate the organizations and dependencies among software components
- A component may be
 - A source code component
 - A run time components or
 - An executable component

Component Diagram



Deploying the System

- 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.

Deployment Diagram

