UML Class Diagram

What is a Class Diagram?

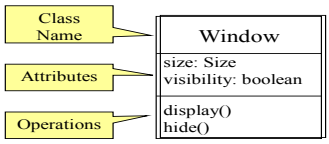
- A Class Diagram is a diagram describing the structure of a system
- shows the system's
  - classes
  - Attributes
  - operations (or methods),
  - Relationships among the classes.

Essential Elements of a UML Class Diagram

- Class
- Attributes
- Operations
- Relationships
  - Associations
  - Generalization
  - Realization
  - Dependency
- Constraint Rules and Notes

Class

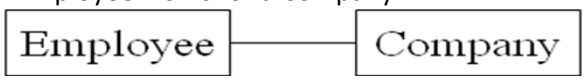
- Describes a set of objects having similar:
  - Attributes (status)
  - Operations (behavior)
  - Relationships with other classes
- Attributes and operations may
  - have their visibility marked:
  - "+" for *public*
  - "#" for *protected*
  - "–" for *private*
  - "~" for *package*



4

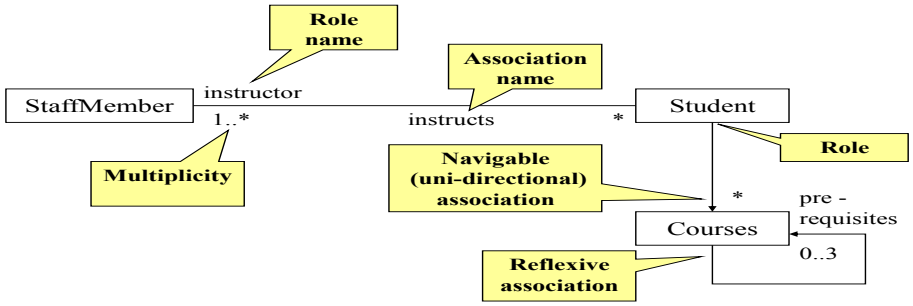
Associations

- An association between two classes indicates that objects at one end of an association "recognize" objects at the other end and may send messages to them.
- Example: "An Employee works for a Company"



- To clarify its meaning, an association may be named.
  - The name is represented as a label placed midway along the association line.
  - Usually a verb or a verb phrase.
- A **role** is an end of an association where it connects to a class.
  - May be named to indicate the role played by the class attached to the end of the association path.
    - Usually a noun or noun phrase
    - Mandatory for reflexive associations

Associations (cont.)



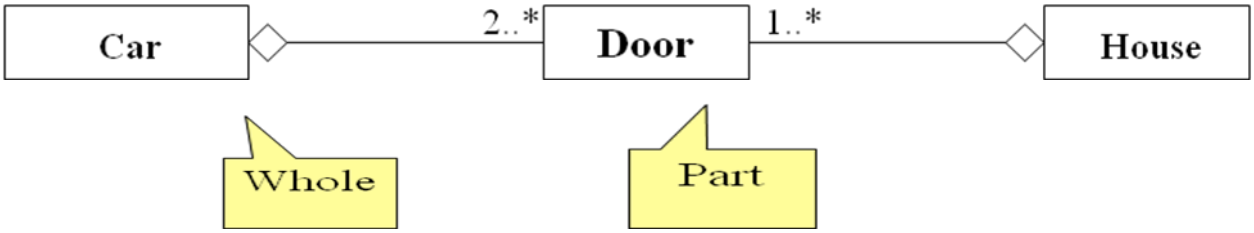
- To clarify its meaning, an association may be named.
  - The name is represented as a label placed midway along the association line.
  - Usually a verb or a verb phrase.
- A **role** is an end of an association where it connects to a class.
  - May be named to indicate the role played by the class attached to the end of the association path.
    - Usually a noun or noun phrase
    - Mandatory for reflexive associations
- Multiplicity
  - the number of objects that participate in the association.
  - Indicates whether or not an association is mandatory.

  Multiplicity Indicators

Exactly one	1
Zero or more (unlimited)	* (0..*)
One or more	1..*
Zero or one (optional association)	0..1
Specified range	2..4
Multiple, disjoint ranges	2, 4..6, 8

Aggregation

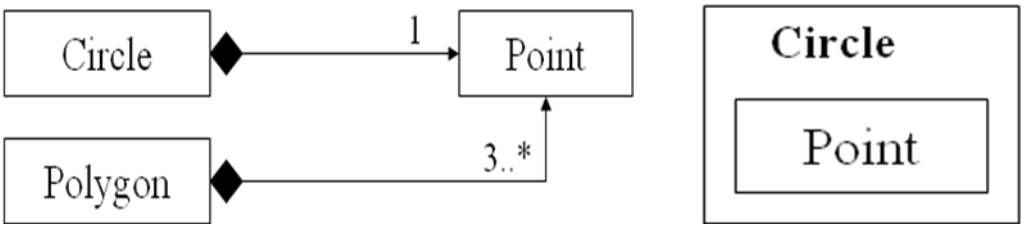
- A special form of association that models a whole-part relationship between an aggregate (the whole) and its parts.
  - Models a "is a part-part of" relationship.



- Aggregation tests:
  - Is the phrase "part of" used to describe the relationship?
    - A door is "part of" a car
  - Are some operations on the whole automatically applied to its parts?
    - Move the car, move the door.
  - Are some attribute values propagated from the whole to all or some of its parts?
    - The car is blue, therefore the door is blue.
  - Is there an intrinsic asymmetry to the relationship where one class is subordinate to the other?
    - A door **is** part of a car. A car **is not** part of a door.

Composition

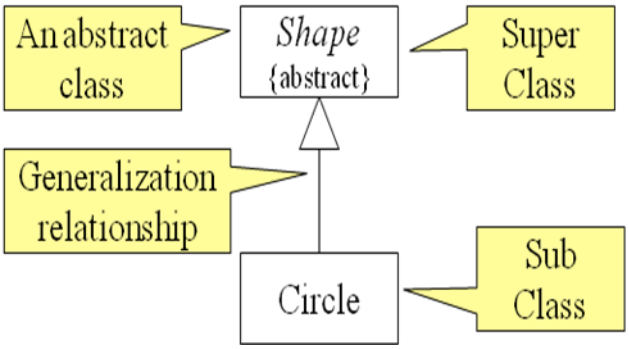
- A strong form of aggregation
  - The whole is the sole owner of its part.
    - The part object may belong to only one whole
  - Multiplicity on the whole side must be zero or one.
  - The life time of the part is dependent upon the whole.
    - The composite must manage the creation and destruction of its parts.



Generalization

- Indicates that objects of the specialized class (subclass) are substitutable for objects of the generalized class (super-class).
  - "is kind of" relationship.

{abstract} is a tagged value that indicates that the class is abstract.
The name of an abstract class should be italicized



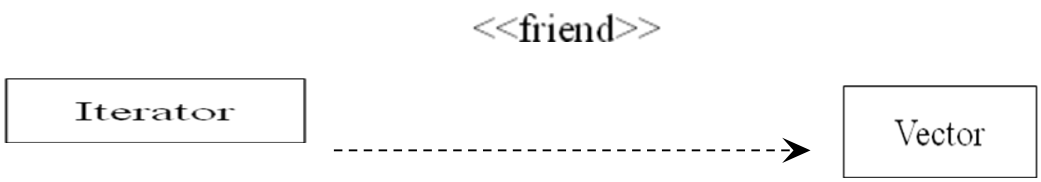
- A sub-class inherits from its super-class
  - Attributes
  - Operations
  - Relationships
- A sub-class may
  - Add attributes and operations
  - Add relationships
  - Refine (override) inherited operations
- A generalization relationship **may not** be used to model interface implementation.

Realization

- A realization relationship indicates that one class implements a behavior specified by another class (an interface or protocol).
- An interface can be realized by many classes.
- A class may realize many interfaces.

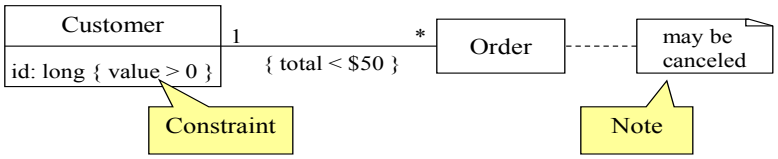
Dependency

- Dependency is a weaker form of relationship which indicates that one class depends on another because it uses it at some point in time.
- One class depends on another if the independent class is a parameter variable or local variable of a method of the dependent class.
- This is different from an association, where an attribute of the dependent class is an instance of the independent class.



Constraint Rules and Notes

- **Constraints** and **notes** annotate among other things associations, attributes, operations and classes.
- Constraints are semantic restrictions noted as Boolean expressions.
  - UML offers many pre-defined constraints.

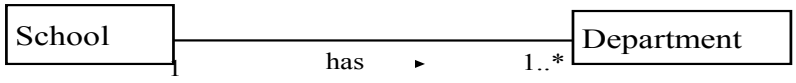


Class Diagram – Example

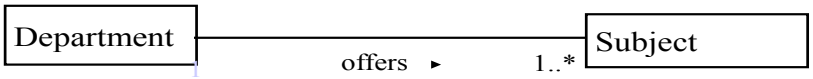
- Draw a class diagram for a information modeling system for a school.
  - School has one or more Departments.
  - Department offers one or more Subjects.
  - A particular subject will be offered by only one department.
  - Department has instructors and instructors can work for one or more departments.
  - Student can enrol in upto 5 subjects in a School.
  - Instructors can teach upto 3 subjects.
  - The same subject can be taught by different instructors.
  - Students can be enrolled in more than one school.

Class Diagram - Example

- School has one or more Departments.

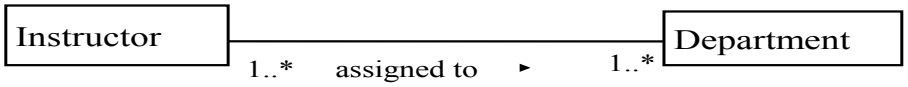


- Department offers one or more Subjects.
- A particular subject will be offered by only one department.

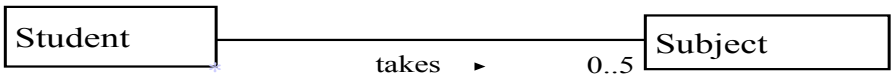


Class Diagram - Example

- Department has Instructors and instructors can work for one or more departments.

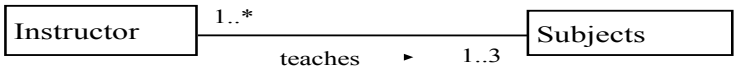


- Student can enroll in upto 5 Subjects.



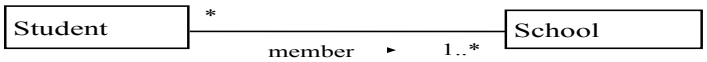
Class Diagram - Example

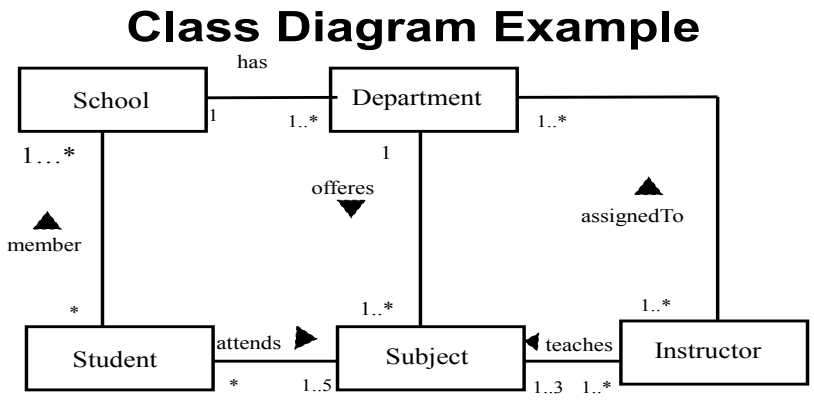
- Instructors can teach up to 3 subjects.
- The same subject can be taught by different instructors.



Class Diagram - Example

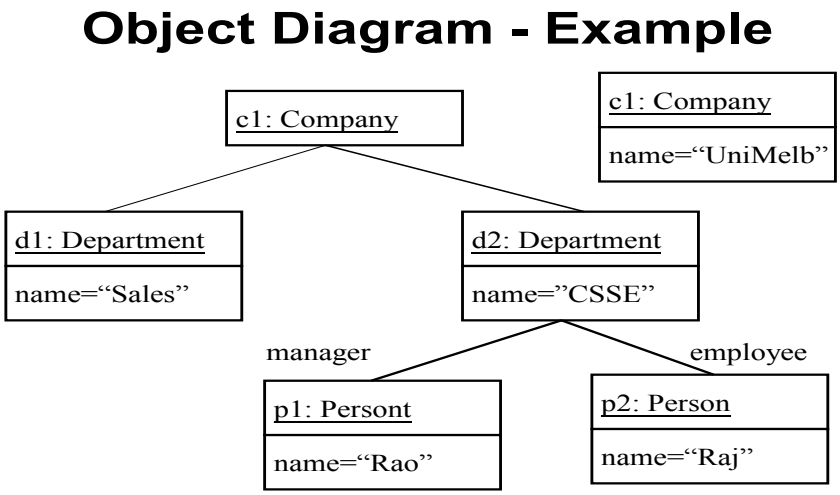
- Students can be enrolled in more than one school.





Object Diagram

- Object Diagram shows the relationship between objects.
- Unlike classes objects have a state.



Summary

- We have discussed the following concepts and UML notations related:
  - Association
  - Generalization
  - Realization
  - Dependency
- How to create a Class Diagram that contains all the above relationships
- Object Diagram for Uni.Dept.system.

Interaction Diagram

Objectives

- Interaction diagrams are created to represent interactive behavior of the system.
  - UML defines 2 types of interaction diagrams

- 1. sequence diagram
- 2. communication diagram

Introduction

Interaction Diagrams are used to model system dynamics

- How do objects change state?
- How do objects interact (message passing)?

Communication & Sequence Diagrams

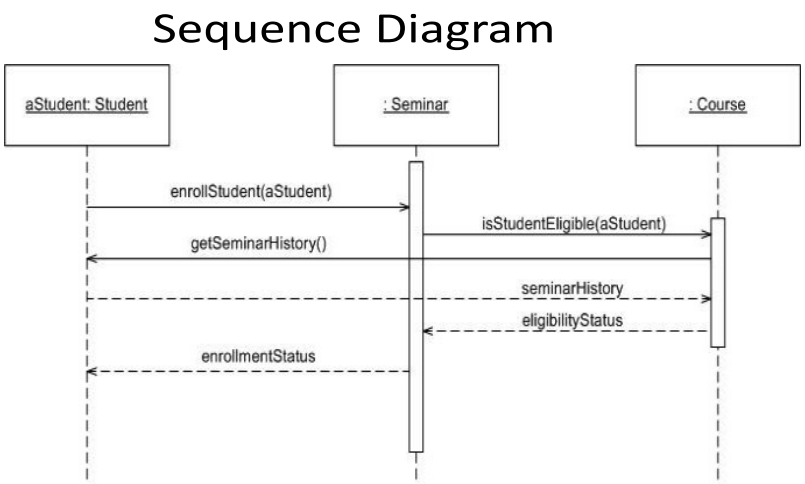
- An Interaction Diagram is a generalization of two specialized UML diagram types
  - Communication Diagrams: Illustrate object interactions organized around the objects and their links to each other
  - Sequence Diagrams: Illustrate object interactions arranged in time sequence
- Both diagram types are semantically equivalent, however, they may not show the same information
  - Communication Diagrams emphasize the structural organization of objects, while Sequence Diagrams emphasize the time ordering of messages
  - Communication Diagrams explicitly show object linkages, while links are implied in Sequence Diagrams

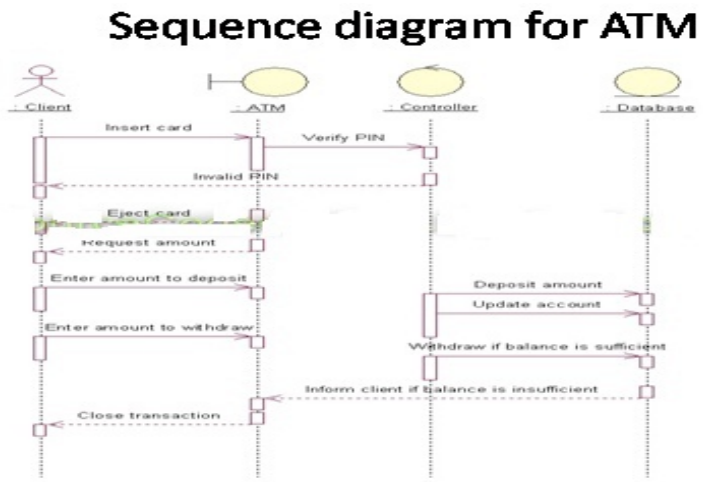
Sequence Diagrams

- Correspond to one scenario within a Use Case
- Model a single operation within a System over time
- Identify the objects involved with each scenario
- Identify the past messages and actions that occur during a scenario
- Identify the required response of each action

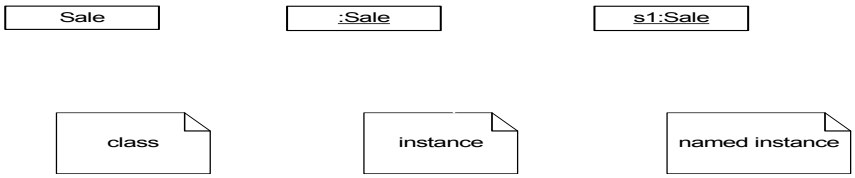
Basic Sequence Diagram Notation

- Links - Sequence Diagrams do not show links
- Message - represented with a message expression on an arrowed line between objects
- Object Lifeline - the vertical dashed line underneath an object
  - Objects do not have a lifeline until they are created
  - The end of an object's life is marked with an "X" at the end of the lifeline
  - Passage of time is from top to bottom of diagram
- Activation - the period of time an object is handling a message (box along lifeline)
  - Activation boxes can be overlaid to depict an object invoking another method on itself
- Conditional Message
  - *[variable = value]* message()
  - Message is sent only if clause evaluates to *true*
- Iteration (Looping)
  - ** [i := 1..N]*: message()
  - "*" is required; [...] clause is optional





Common Interaction Diagram Notation



Communication Diagrams

- Objects are connected with numbered (sequenced) arrows along links to depict information flow.
- Arrows are drawn from the interaction source.
- The object pointed to by the arrow is referred to as the target.
- Arrows are numbered to depict their usage order within the scenario.
- Arrows are labeled with the passed message.

Basic Communication Diagram Notation

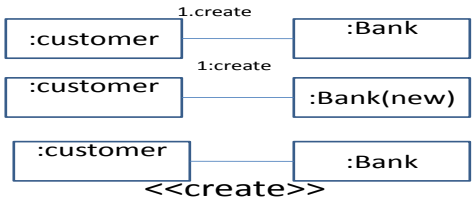
- Link - connection path between two objects (an instance of an association)

- Message - represented with a message expression on an arrowed line between objects
- Sequence Number - represents the order in which the flows are used
- Conditional Message
  - Seq. Number *[variable = value]* : message()
  - Message is sent only if clause evaluates to *true*
- Iteration (Looping)
  - Seq. Number ** [i := 1..N]*: message()
  - "*" is required; [...] clause is optional

- Self or this : the message can be set for the object itself.

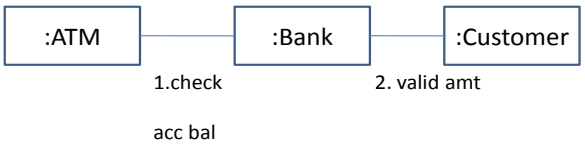
Creation of instances

- Any message can be used to create instance.
- Ex create message which creates an instance.
- Parameters can be passed to create message.
- <<create>> used as a stereotype.



Message numbering scheme

- 1. the first message must not be numbered.
- 2. the order and nesting of message is shown with legal numbering.
- Nesting can be denoted by a perpendicular incoming message number to outgoing message.

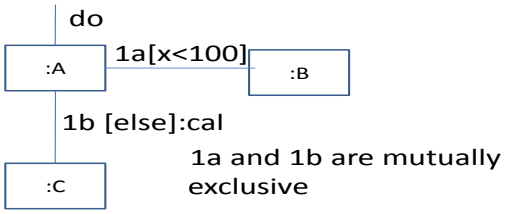


Conditional message

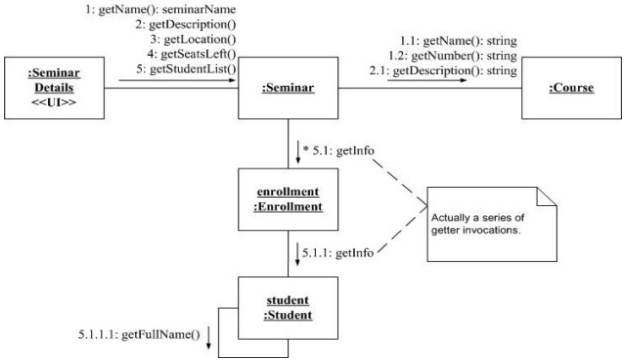
- It is shown in a square bracket.
- The message is sent only when the condition is true.

Mutually exclusive conditional paths

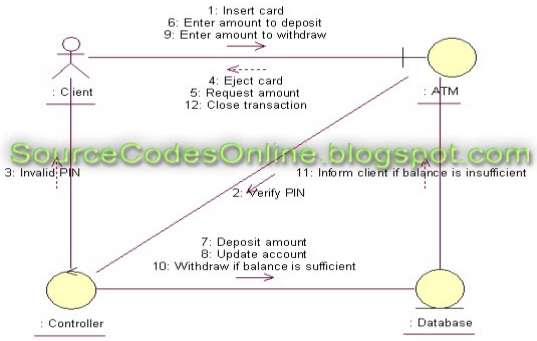
- The conditional expression can be modified using the conditional path.



Communication Diagram



Communication diagram ATM



Interaction Diagram Strengths

- Communication Diagram
  - Space Economical - flexibility to add new objects in two dimensions

  - Better to illustrate complex branching, iteration, and concurrent behavior
- Sequence Diagram
  - Clearly shows sequence or time ordering of messages
  - Simple notation

Interaction Diagram Weaknesses

- Communication Diagram
  - Difficult to see sequence of messages
  - More complex notation
- Sequence Diagram
  - Forced to extend to the right when adding new objects; consumes horizontal space

Conclusions

- Beginners in UML often emphasize Class Diagrams. Interaction Diagrams usually deserve more attention.
- There is no rule about which diagram to use. Both are often used to emphasize the flexibility in choice and to reinforce the logic of the operation. Some tools can convert one to the other automatically.

UML State Machine Diagrams and Modeling

- These diagrams illustrate the interesting events and states of an object and the behavior of an object in reaction to an event.
- The state diagram shows the life cycle of the object.
- Sometimes it is not possible to show each and every event in the state diagram.

Events, states and transitions

- Event: significant or noteworthy occurrence of something..
  - E.g., customer inserts ATM card.
- State: it is a condition of an object at a moment in time (between events).

ex waiting for user to enter PIN

authenticating the user

Dispensing cash (these states represent the object is waiting for something or doing something.)

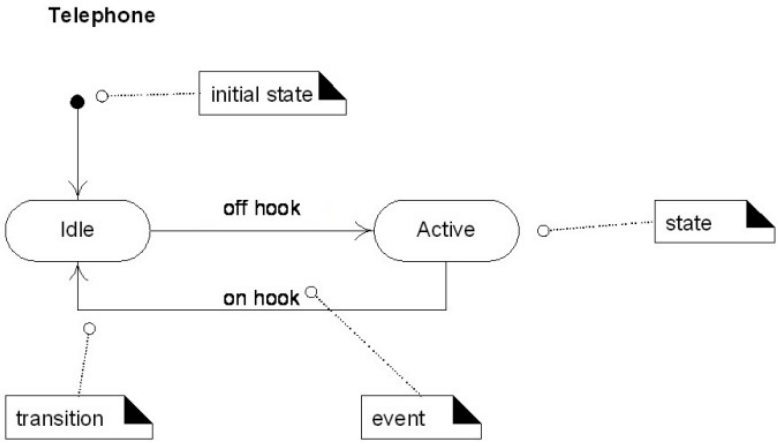
- Transition: a relationship between two states; when an event occurs, the object moves from the current state to a related state

ex when a system performs “startup” the transition from *ideal* state to *active* state.

UML State Machine Diagram

- States shown as rounded rectangles.
- Transitions shown as arrows.
- Events shown as labels on transition arrows.
- Initial pseudo-state automatically transitions to a particular state on object instantiation.
- Events with no corresponding transitions are ignored.

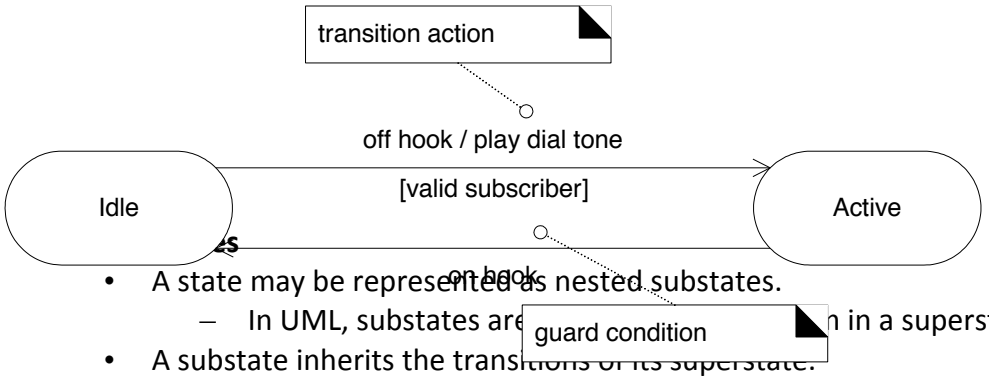
State machine diagram for a telephone



Transition Actions and Guards

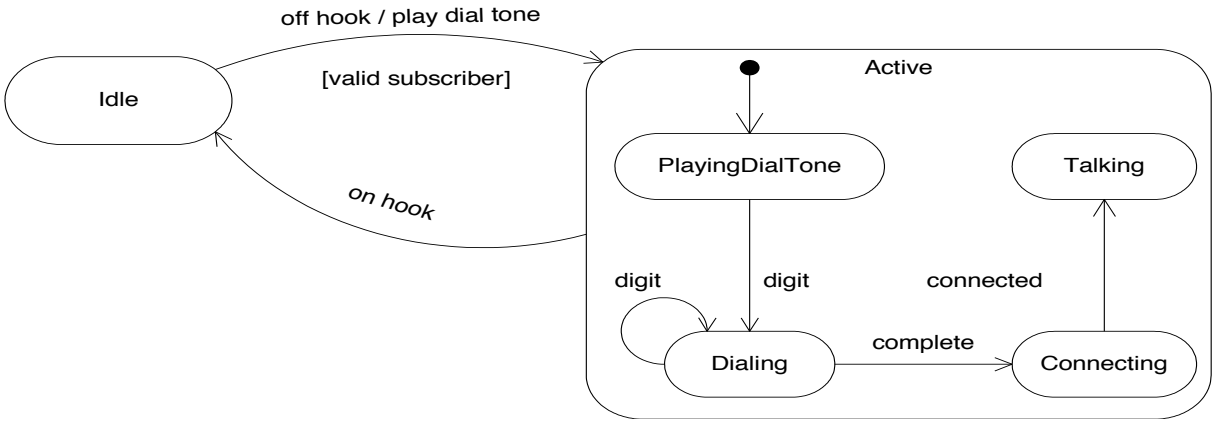
- A transition can cause an action to fire.
  - In software implementation, a method of the class of the state machine is invoked.
- A transition may have a conditional guard.
  - The transition occurs only if the test passes.

Transition action and guard notation



- A state may be represented as nested substates.
  - In UML, substates are shown in a superstate box.
- A substate inherits the transitions of its superstate.

Allows succinct state machine diagrams.

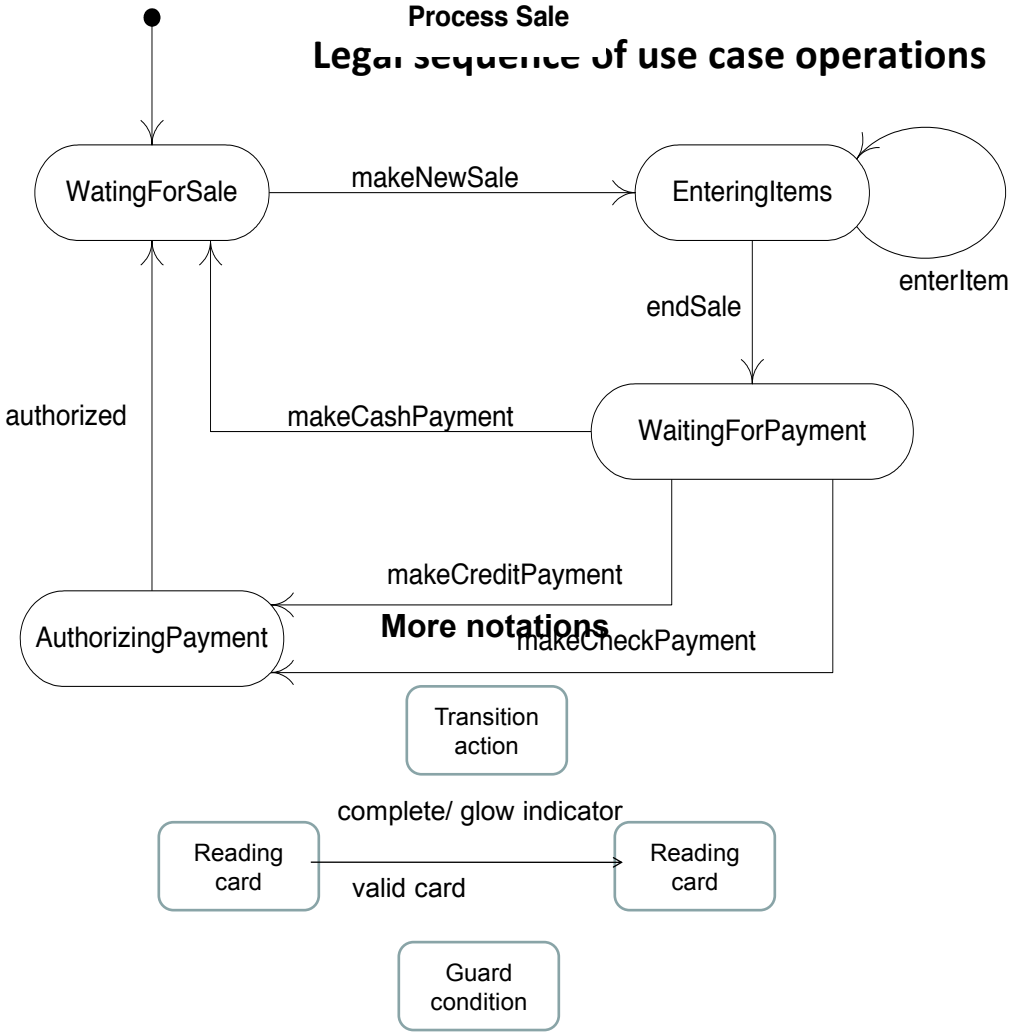


State-Independent

- State-independent (modeless) — type of object that always responds the same way to an event.
- Ex if an object receives the message “customer pressed the cancel button” or “customer finished the transaction” , the corresponding method will be **Ejecting card.**

State-Dependent

- State-dependent (modal) — type of object that reacts differently to events depending on its state or mode.
- Ex for an “insert card” event the state will be “reading card” whereas “enter PIN” event the state will be “validating PIN”
-



The glow indicator denotes the transition action and the valid card denotes guard condition.

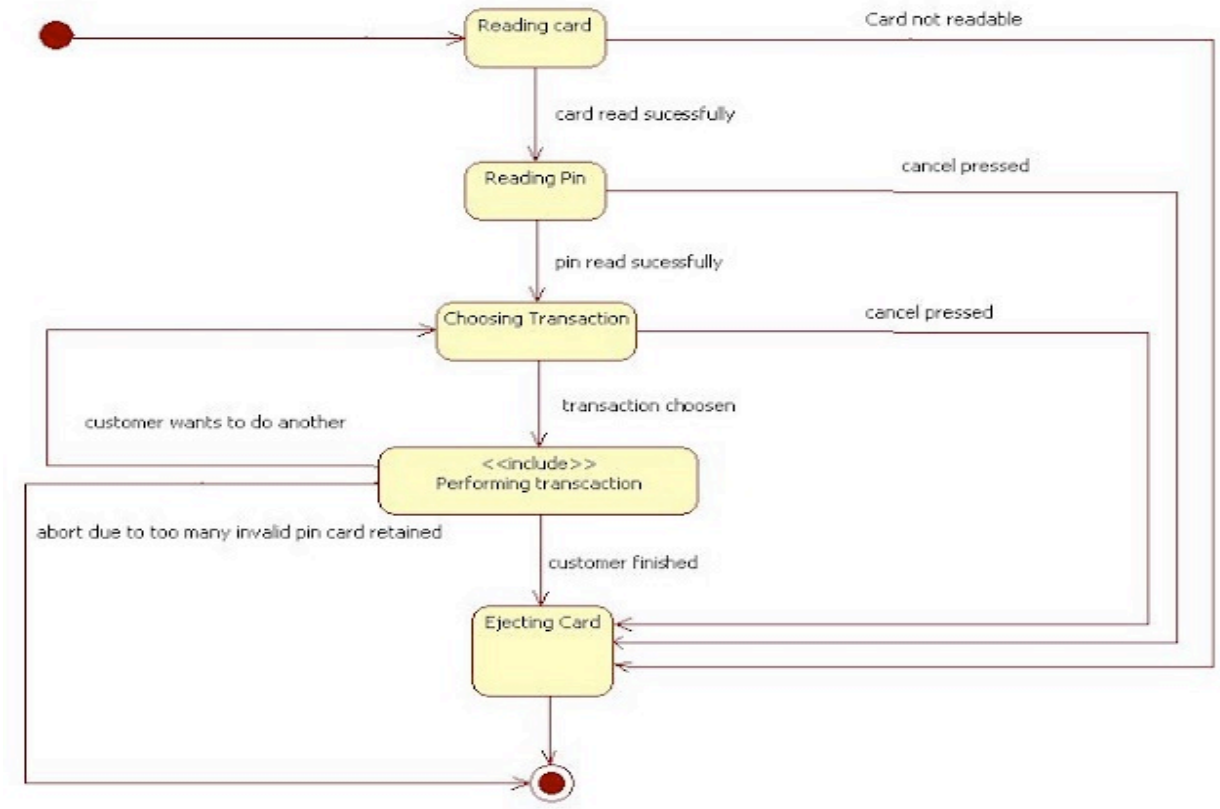
Nested states

- Nested states are the states that occur within the one super state.
- Ex Active state is a super state in which the sub states ***display, reading card, validating PIN*** are present.
- When the ***perform shutdown*** event occurs the system is restored to idle to ***active state***.

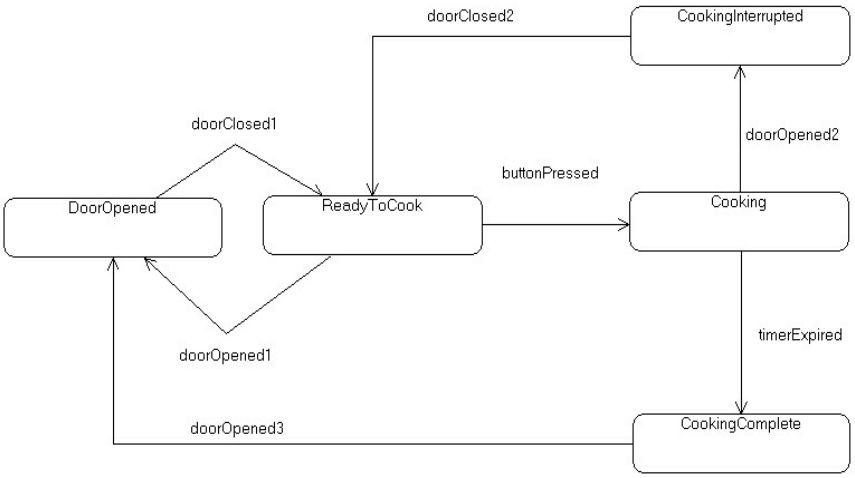
State machine model - State machine model -

State machine model -

ATM



Microwave cooking state diagram

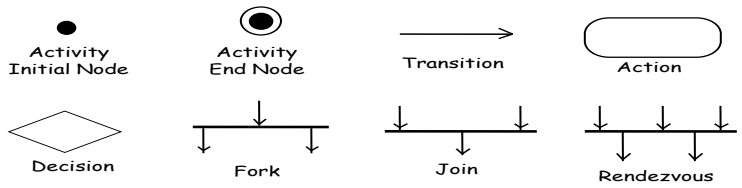


Activity Diagram

- Similar to flow chart that describes sequence of activities.
- Useful in:
  - Business Modelling (business workflow).
  - Use Cases (interrelation and interaction).
  - Design (algorithm, complex sequence etc).
- Often associated with several classes.
- One of the strengths of activity diagrams is the representation of *concurrent* activities.

Activity Diagram

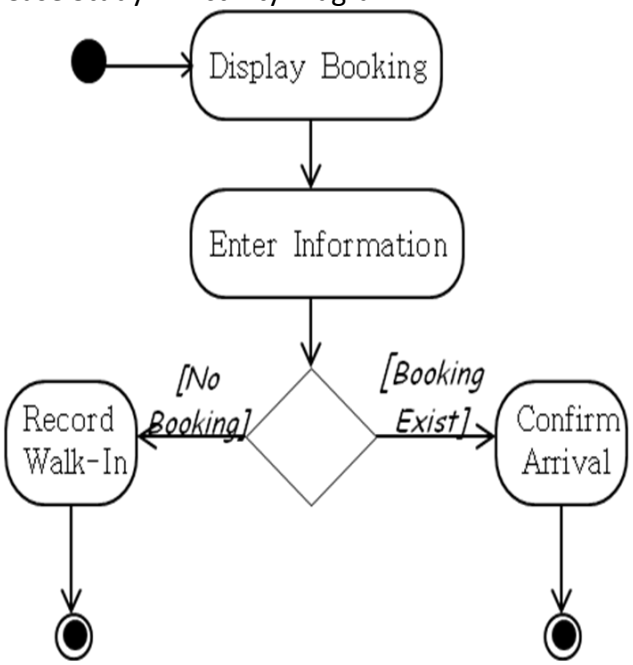
- Diagram Elements:



2

- **Action:**
  - Fundamental block in an activity diagram.
  - Represents a unit of work (something is done).
  - Automatic transition upon completion.
- **Transition:**
  - Represents the control flow: it is simply a movement between *actions.*
- **Initial and End Node:**
  - Show the beginning and ending points in an activity diagram.

Case Study 1: Activity Diagram



- Documenting the sequence when customer arrives.
- Note the interrelation of *Record Arrival* and *Record Walk-In.*

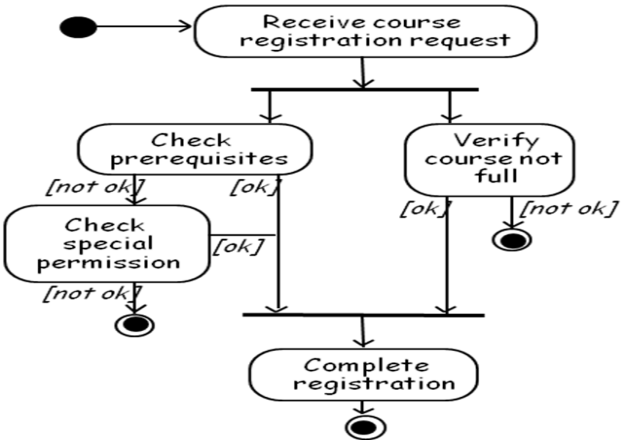
Concurrent Actions

- Independent actions which can be carried out at the same time (in parallel).
- Shown using *forks, joins* and *rendezvous*:
  - **Fork:**
    - One incoming transition and multiple outgoing transitions.
    - Execution splits into multiple concurrent threads.
  - **Rendezvous:**
    - Multiple incoming and multiple outgoing transitions.
    - All incoming transitions must occur before the outgoing transitions.

Representing concurrency

- **Join:**
  - Multiple incoming transitions and one outgoing transition.
  - The outgoing transition will be taken when all incoming transitions have occurred.

Activity Diagram: Another Example

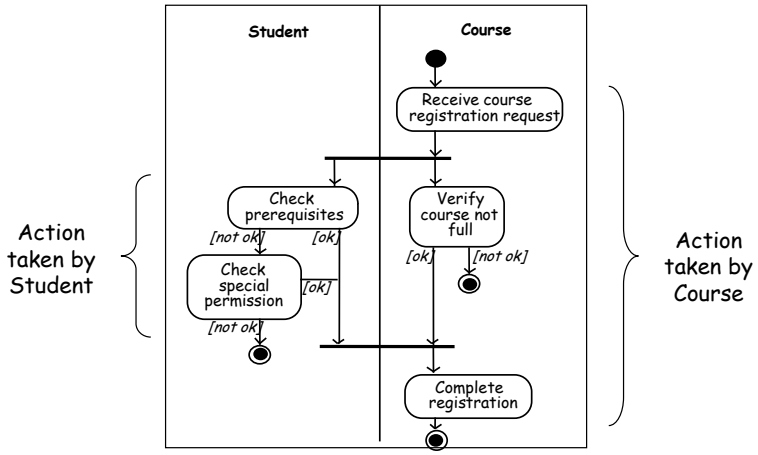


- Use Cases:
  - Submit registration request.
- Register student:
  - Verify prerequisites.
  - Verify course enrolment.
  - Verify special cases.

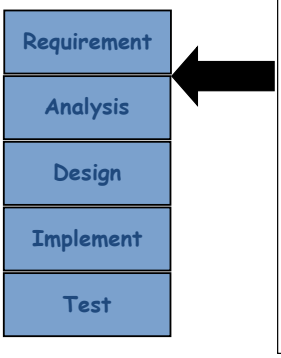
Swimlanes

- Activity diagrams are most often associated with several classes.
  - The partition of activities among the existing classes can be explicitly shown using *swimlanes*.
  - Further clarify *where* an action takes place.

Swimlanes: Example



Where are we now?



- Typical Artifacts:
  - Requirement Specification.
  - Use Cases:
    - Use Case Description.
    - Use Case Diagrams.
    - Activity Diagrams.
    - Glossary.
  - Domain Model.

Other UML Notations

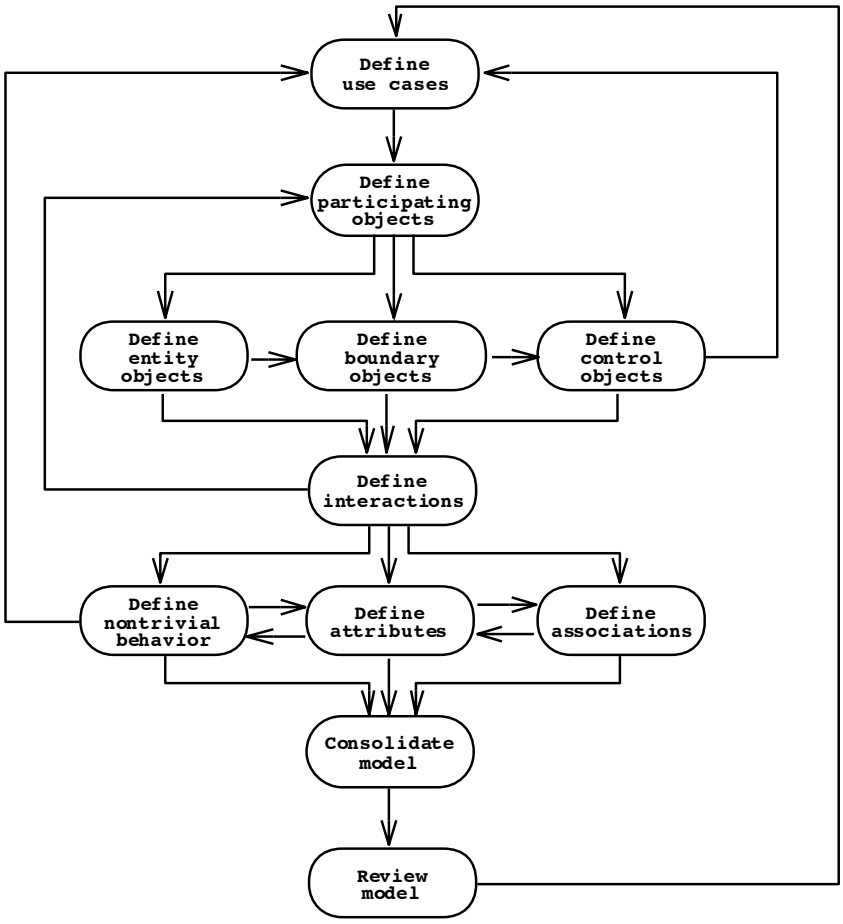
UML provide other notations that we will be introduced in subsequent lectures, as needed.

- Implementation diagrams
  - Component diagrams
  - Deployment diagrams
  - Introduced in lecture on System Design
- Object Constraint Language (OCL)
  - Introduced in lecture on Object Design

Summary

- UML provides a wide variety of notations for representing many aspects of software development
  - Powerful, but complex language
  - Can be misused to generate unreadable models
  - Can be misunderstood when using too many exotic features
- We concentrate only on a few notations:
  - Functional model: use case diagram
  - Object model: class diagram
  - Dynamic model: sequence diagrams, statechart and activity diagrams

Analysis: UML Activity Diagram



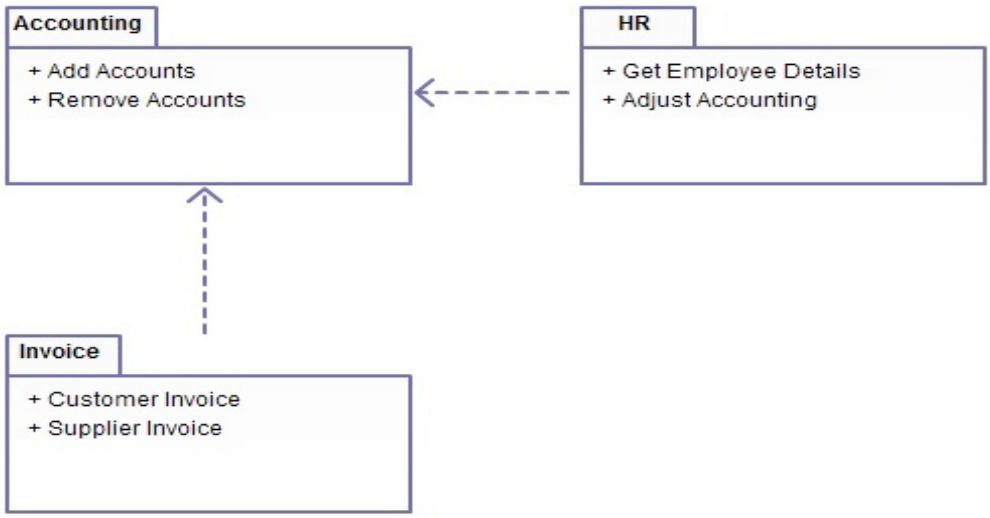
Package Diagrams

- A package diagram is a UML diagram composed only of packages and the dependencies between them.
- A package is a UML construct that enable to organize model elements, such as use cases or classes, into groups.
- Packages are depicted as file folders and can be applied on any UML diagram.
- Create a package diagram to: Depict a high-level overview of the requirements (over viewing a collection of UML Use Case diagrams)
- Depict a high-level overview of the architecture/design (over viewing a collection of UML Class diagrams).

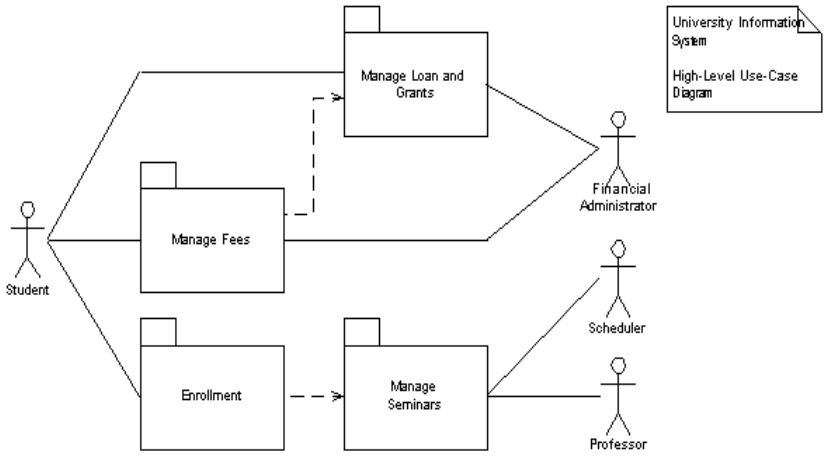
General Guidelines

- Give Packages Simple, Descriptive Names
- Apply Packages to Simplify Diagrams
- Packages Should be Cohesive
- Indicate Architectural Layers With Stereotypes on Packages
- Avoid Cyclic Dependencies Between Packages

- Package Dependencies Should Reflect Internal Relationships
  Class Package Example

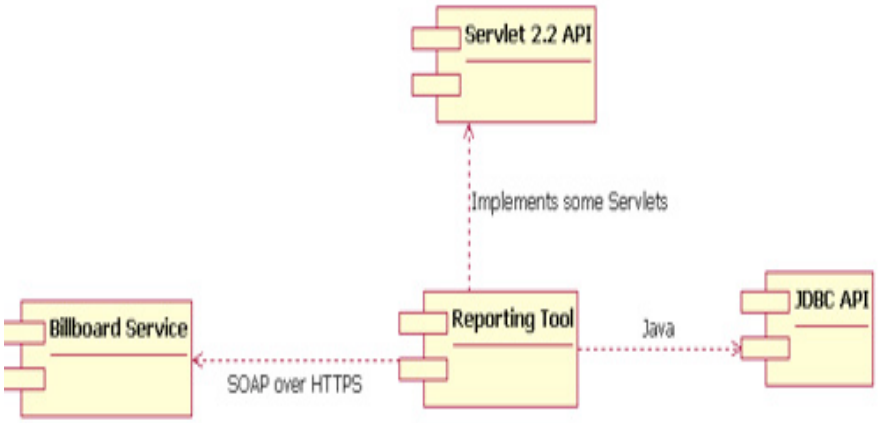


Use Case Package Example

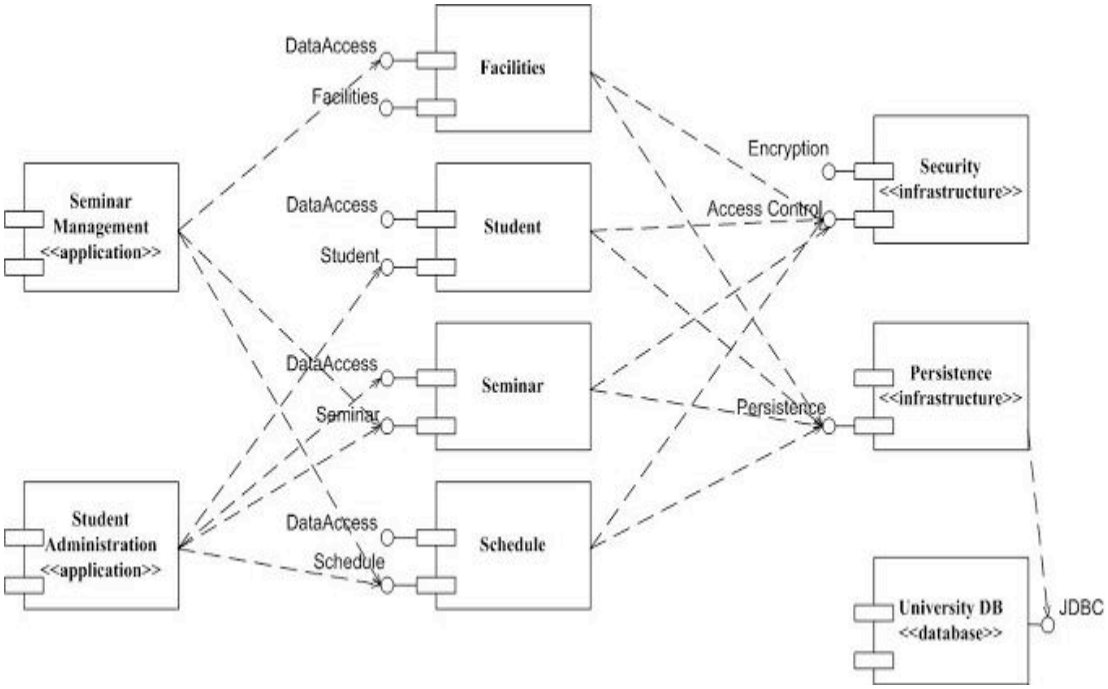


Component and Deployment Diagrams
What is a Component Diagram

- The component diagram's main purpose is to show the structural relationships between the components of a system.
- UML component diagrams are to model the high-level software components, and more importantly the interfaces to those components.

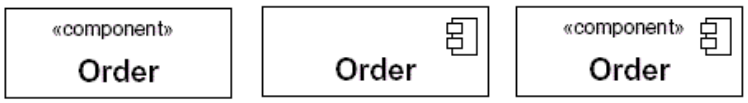


Component Diagram Example in UML



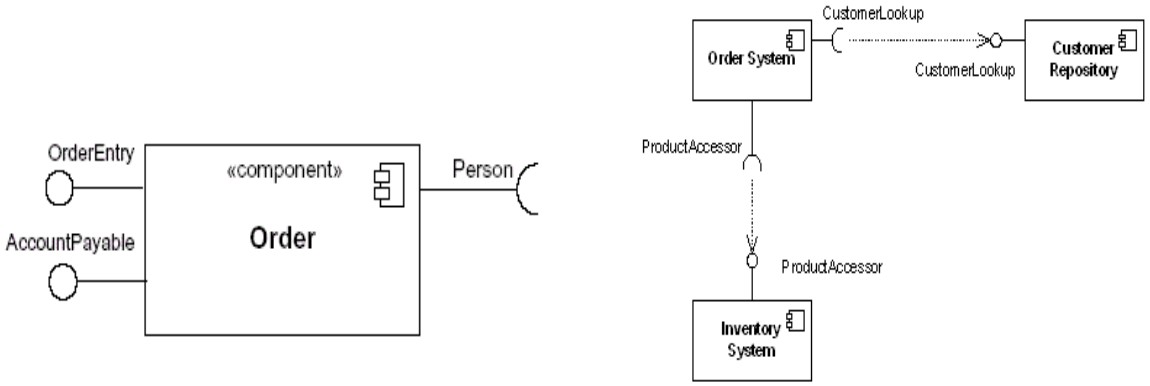
Basic Notation Used in Component Diagrams

- Drawing a component in UML is now very similar to drawing a class on a class diagram.
- In UML , a component is drawn as a rectangle with optional compartments stacked vertically.
- A high-level, abstracted view of a component in UML can be modeled as just a rectangle with the component's name and the component stereotype text and/or icon.



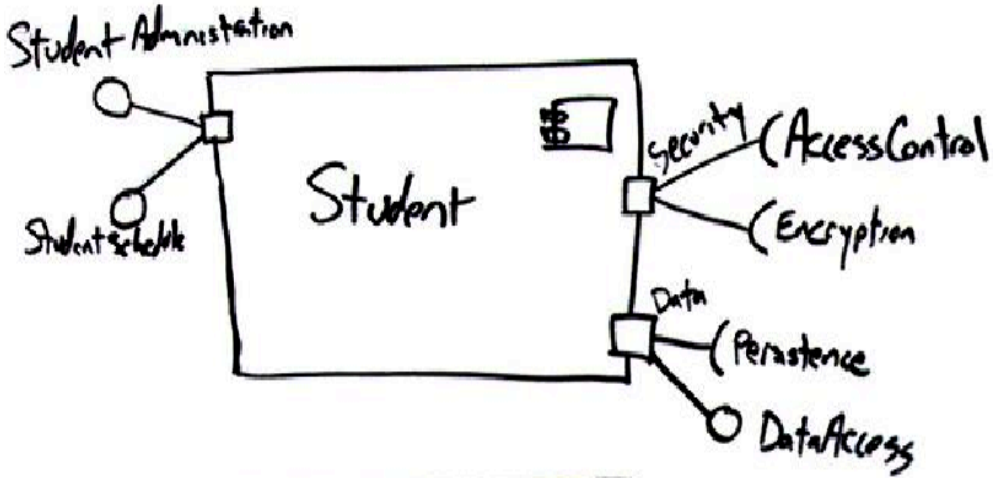
Modeling a Component's Interfaces

- UML 2 has also introduced another way to show a component's provided and required interfaces. In the first way builds off the single rectangle, with the component's name in it, and places interface symbols, connected to the outside of the rectangle.
- In this second approach the interface symbols with a complete circle at their end represent an interface that the component provides -- this lollipop symbol is shorthand for a realization relationship of an interface classifier.



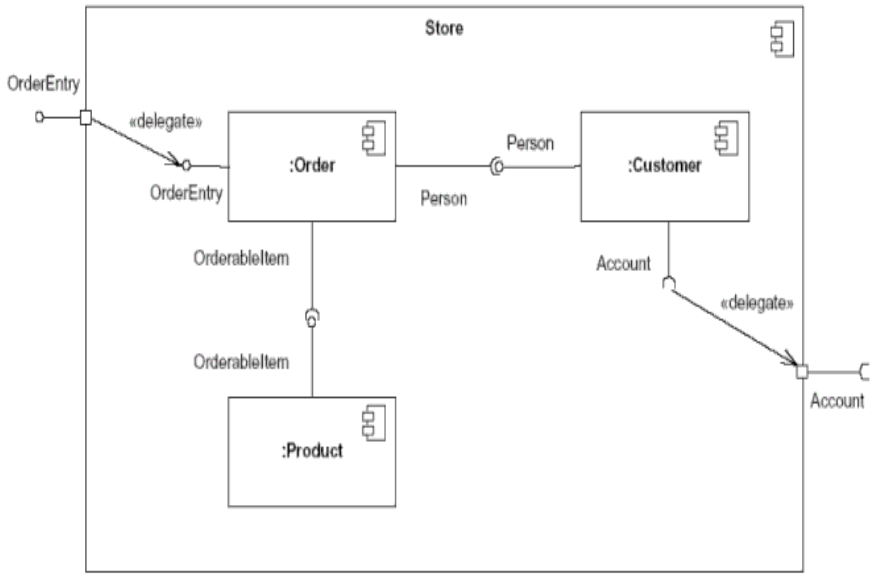
Interfaces and Ports

- Components may both provide and require interfaces.
- An interface is the definition of a collection of one or more methods, and zero or more attributes, ideally one that defines a cohesive set of behaviors.
- A port is a feature of a classifier that specifies a distinct interaction point between the classifier and its environment. Ports are depicted as small squares on the sides of classifiers.
- Ports can be named, such as the Security and Data ports on the Student component.
- Ports can support unidirectional communication or bi-directional communication.
- The Student component implements three ports, two unidirectional ports and one bi-directional ports. The left-most port is an input port, the Security port is an output port, and the Data port is a bi-directional port.



Beyond the Basics

- The component diagram is one of the easier-to-understand diagrams.
- To show a component's inner structure, merely draw the component larger than normal and place the inner parts inside the name compartment of the encompassing component.
- Following fig shows the Store's component inner structure.

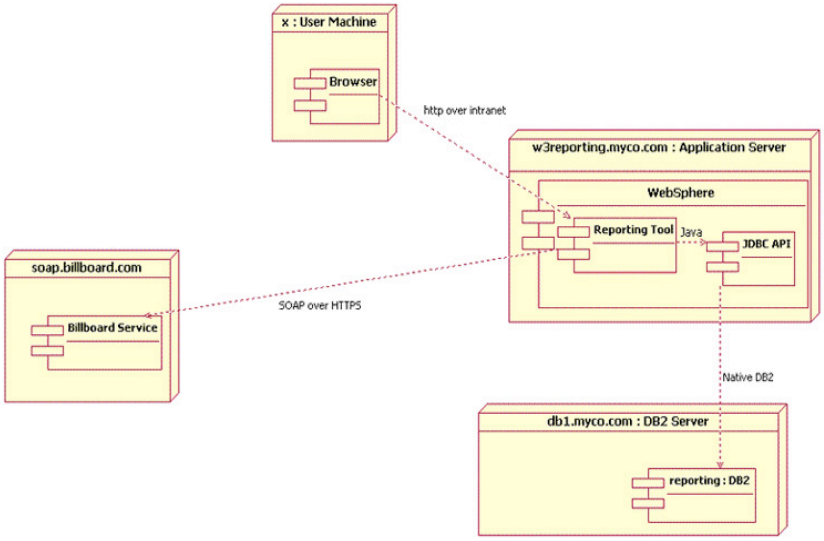


What is a Deployment Diagram?

- The deployment diagram shows how a system will be physically deployed in the hardware environment.
- Its purpose is to show where the different components of the system will physically run and how they will communicate with each other.

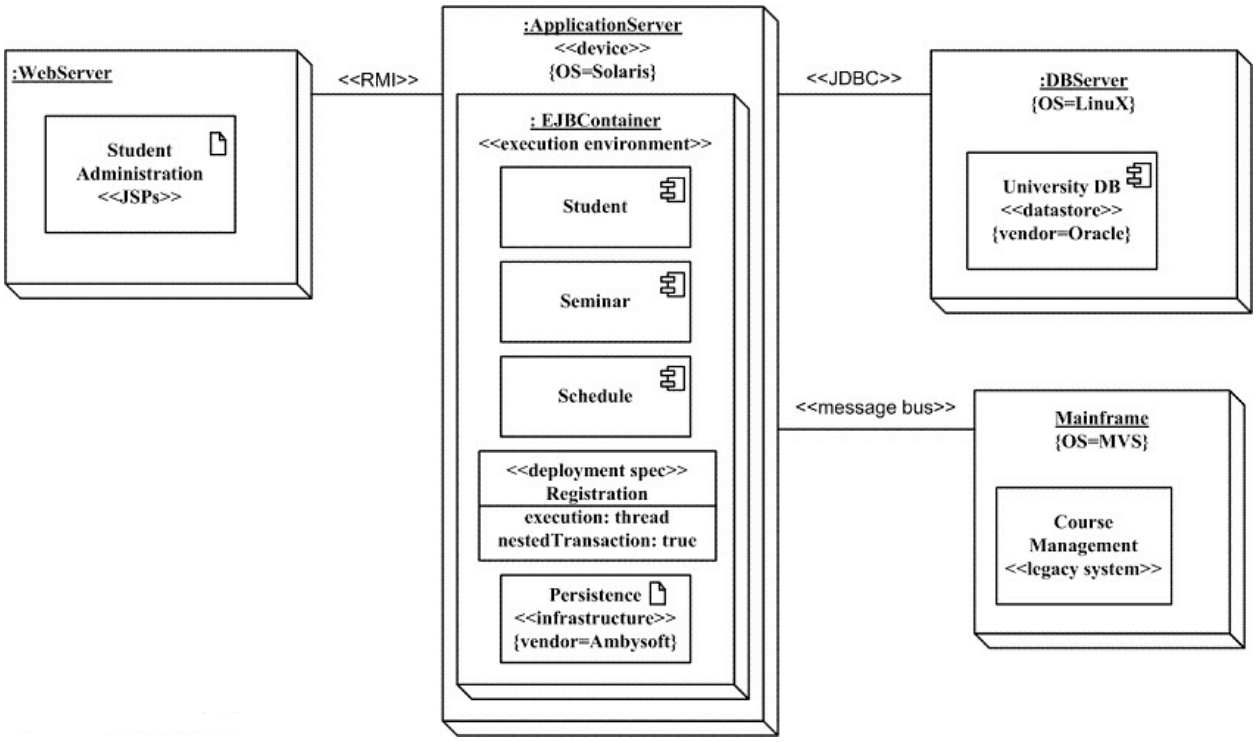
- The notation in a deployment diagram includes the notation elements used in a component diagram, with a couple of additions, including the concept of a node.
- A node represents either a physical machine or a virtual machine node (e.g., a mainframe node).
- To model a node, simply draw a three-dimensional cube with the name of the node at the top of the cube. Use the naming convention used in sequence diagrams.

Deployment Diagram Example



UML Deployment Diagram Example

- The three-dimensional boxes represent nodes, either software or hardware. Physical nodes should be labeled with the stereotype *device*, to indicate that it is a physical device such as a computer or switch.
- Connections between nodes are represented with simple lines, and are assigned stereotypes such as *RMI* and *message bus* to indicate the type of connection.


:WebServer
Student Administration
<<JSPs>>
<<RMI>>
:ApplicationServer
<<device>>
{OS=Solaris}
: EJBContainer
<<execution environment>>
Student
Seminar
Schedule
<<deployment spec>>
Registration
execution: thread
nestedTransaction: true
Persistence
<<infrastructure>>
{vendor=Ambysoft}
<<JDBC>>
:DBServer
{OS=LinuX}
University DB
<<datastore>>
{vendor=Oracle}
<<message bus>>
Mainframe
{OS=MVS}
Course Management
<<legacy system>>