

# UML TUTORIAL

Md. Shafiuzzaman

# Definition of UML

Unified Modeling Language (UML) is a standardized (ISO/IEC 19501:2005), general-purpose modeling language in the field of software engineering. The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.



# History of UML



- The Unified Modeling Language was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in the 1990s.
- It was adopted by the Object Management Group (OMG) in 1997, and has been managed by this organization ever since.
- In 2000 the Unified Modeling Language was accepted by the International Organization for Standardization (ISO) as industry standard for modeling software-intensive systems.
- The current version of the UML is 2.4.1 published by the OMG in August 2011.

# Types of UML Diagram



- **Structure diagrams** (emphasize the things that must be present in the system being modeled)
- **Behavior diagrams** (emphasize what must happen in the system being modeled.)

# Behavior Diagrams



- Activity Diagram
- State Machine Diagram
- Use Case Diagram
- Communication Diagram
- Interaction Overview Diagram
- Sequence Diagram
- Timing Diagram

# Structure Diagrams



- Class Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram

# Activity Diagram



- Activity diagrams model the behaviors of a system, and the way in which these behaviors are related in an overall flow of the system.
- UML activity diagrams are the object-oriented equivalent of flow charts and data-flow diagrams from structured development

# Purpose of Activity Diagram

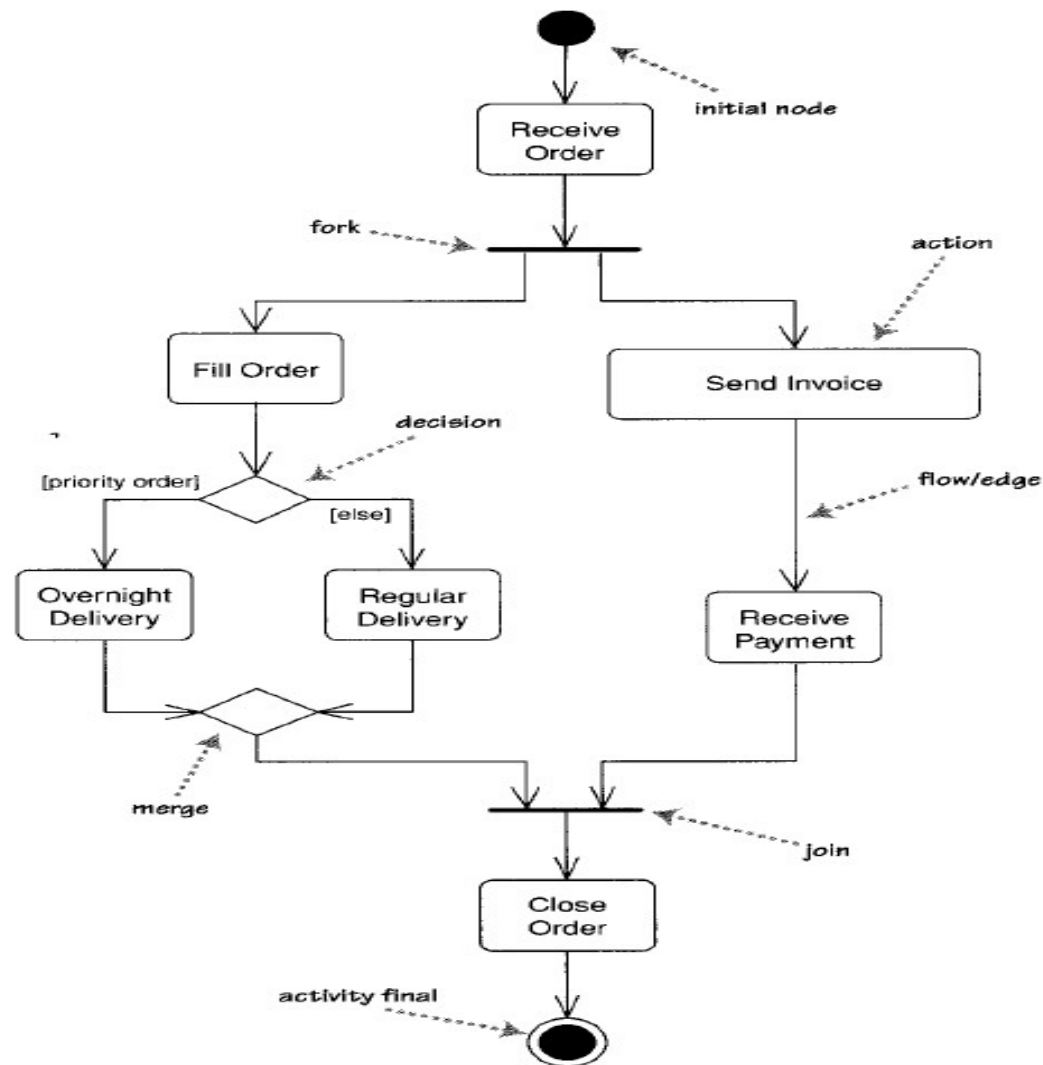


Activity diagrams are used to explore the logic of

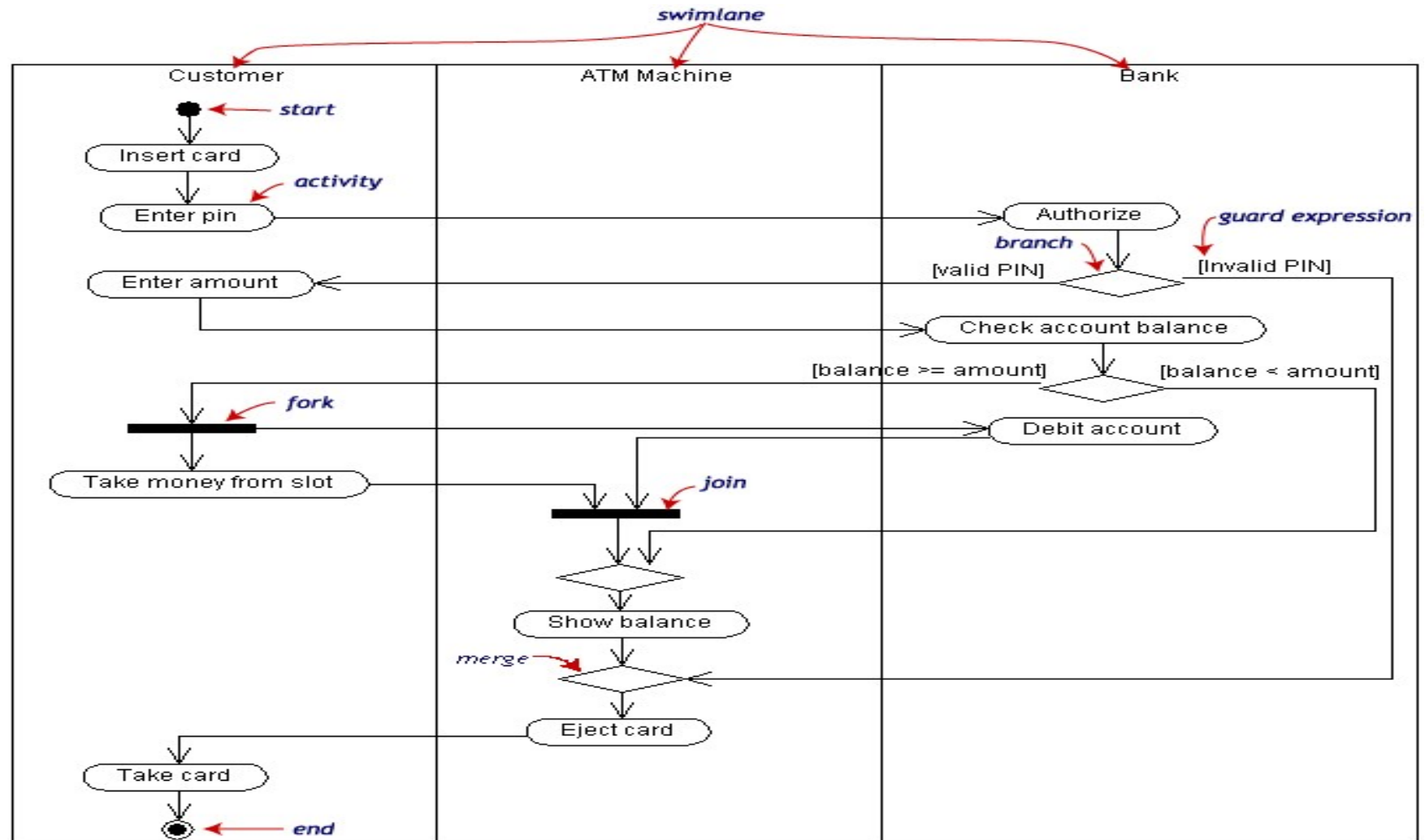
- a complex operation,
- a complex business rule,
- a single use case,
- several use cases,
- a business process,
- concurrent processes,
- software processes.



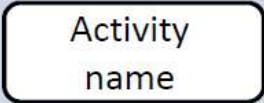

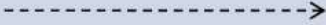


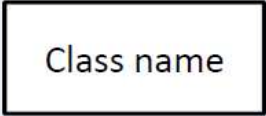
# Example Activity Diagram 1



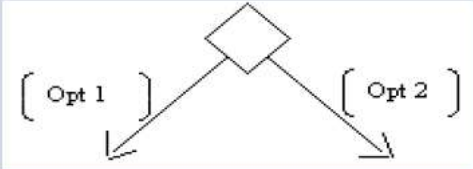
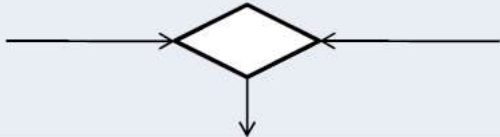
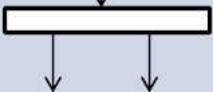
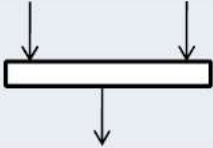

# Example Activity Diagram 2



# Elements of Activity Diagram

| Description  | Symbol  |
|--|---|
| <b>Activity</b> : Is used to represent a set of actions  |    |
| <b>A Control Flow</b> : Shows the sequence of execution  |    |
| <b>An Object Flow</b> : Shows the flow of an object from one activity (or action) to another activity (or action). |    |
| <b>An Initial Node</b> : Portrays the beginning of a set of actions or activities                                  |   |
| <b>A Final-Activity Node</b> : Is used to stop all control flows and object flows in an activity (or action)       |  |
| <b>An Object Node</b> : Is used to represent an object that is connected to a set of Object Flows.                 |  |

# Elements of Activity Diagram

| Description  | symbol  |
|--|---|
| A Decision Node: Is used to represent a test condition to ensure that the control flow or object flow only goes down one path                    |    |
| A Merge Node: Is used to bring back together different decision paths that were created using a decision-node.                                   |    |
| A Fork Node: Is used to split behavior into a set of parallel or concurrent flows of activities (or actions)                                     |   |
| A Join Node: Is used to bring back together a set of parallel or concurrent flows of activities (or actions).                                    |  |
| A Swimlane :A swimlane is a way to group activities performed by the same actor on an activity diagram or to group activities in a single thread |  |

# Practice Activity Diagram

Draw an activity diagram for the following problem:  
Appointment system for doctor office.

1. A patient came to office, the scheduler get patient info.
2. If the patient is new the scheduler make new patient record.
3. The scheduler display list of possible appointments to patient.
4. Patient choose new appointments , modify appointments or cancel his appointments .
5. Patient make payment.

# Use Case Diagram

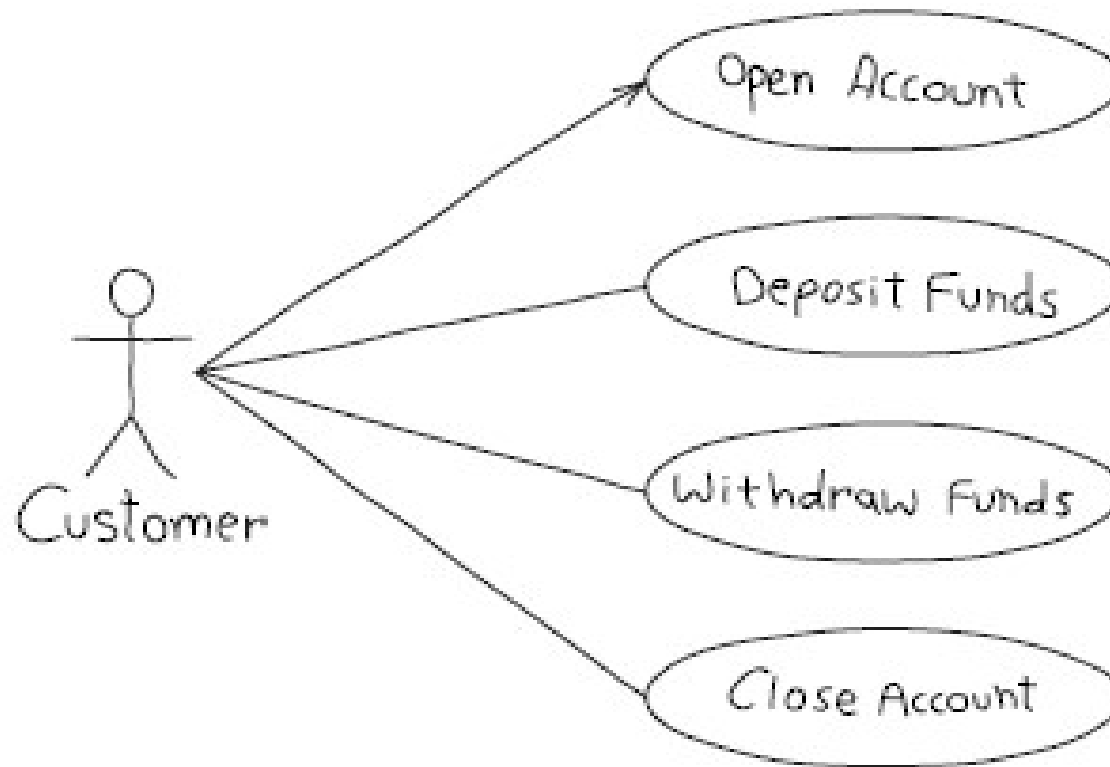
- Use Case diagrams capture Use Cases and the relationships between Actors and the subject (system).

# Purpose of Use Case Diagram



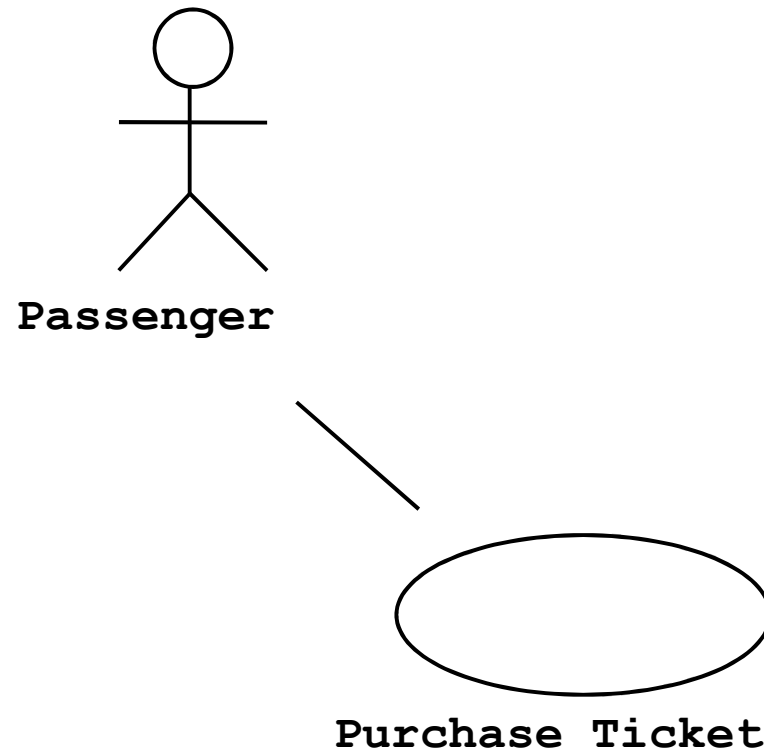
- Describe the functional requirements of the system
- Describe the manner in which outside things (Actors) interact at the system boundary
- Describe the response of the system.

# Example Use Case Diagram 1

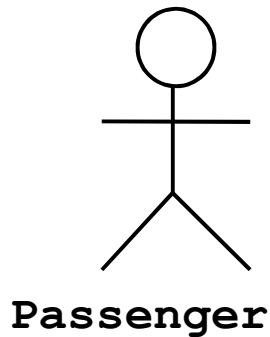




# Example Use Case Diagram 2



# Actors



An actor models an external entity which communicates with the system:

User

External system

Physical environment

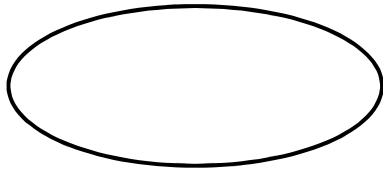
An actor has a unique name and an optional description.

Examples:

Passenger: A person in the train

GPS satellite: Provides the system with  
GPS coordinates

# Use Case



**PurchaseTicket**

A use case represents a class of functionality provided by the system as an event flow.

A use case consists of:

Unique name

Participating actors

Entry conditions

Flow of events

Exit conditions

Special requirements

# Use Case Example

*Name:* Purchase ticket

*Participating actor:* Passenger

*Entry condition:*

- Passenger standing in front of ticket distributor.
- Passenger has sufficient money to purchase ticket.

*Exit condition:*

- Passenger has ticket.

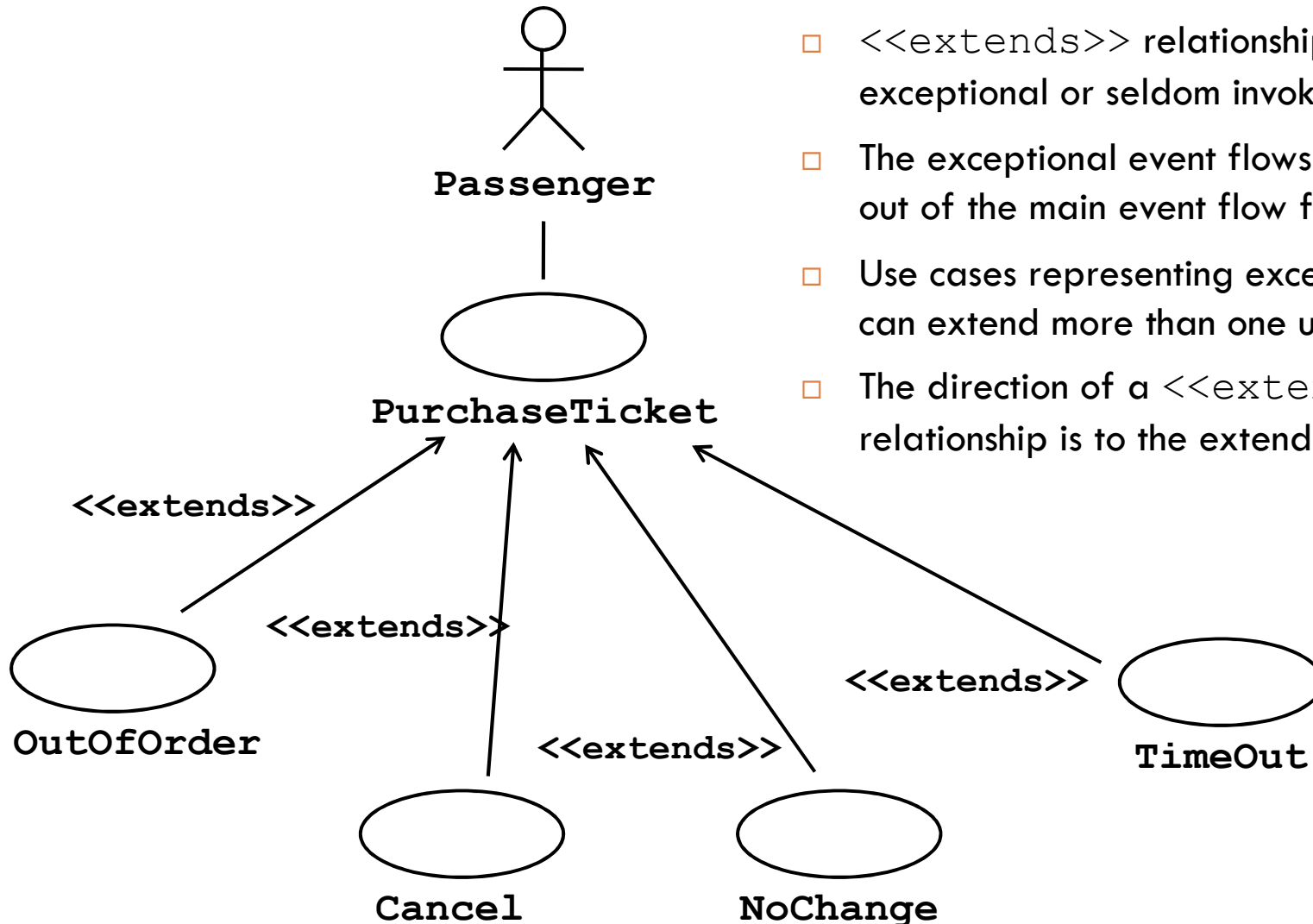
*Event flow:*

1. Passenger selects the number of zones to be traveled.
2. Distributor displays the amount due.
3. Passenger inserts money, of at least the amount due.
4. Distributor returns change.
5. Distributor issues ticket.

Anything missing?

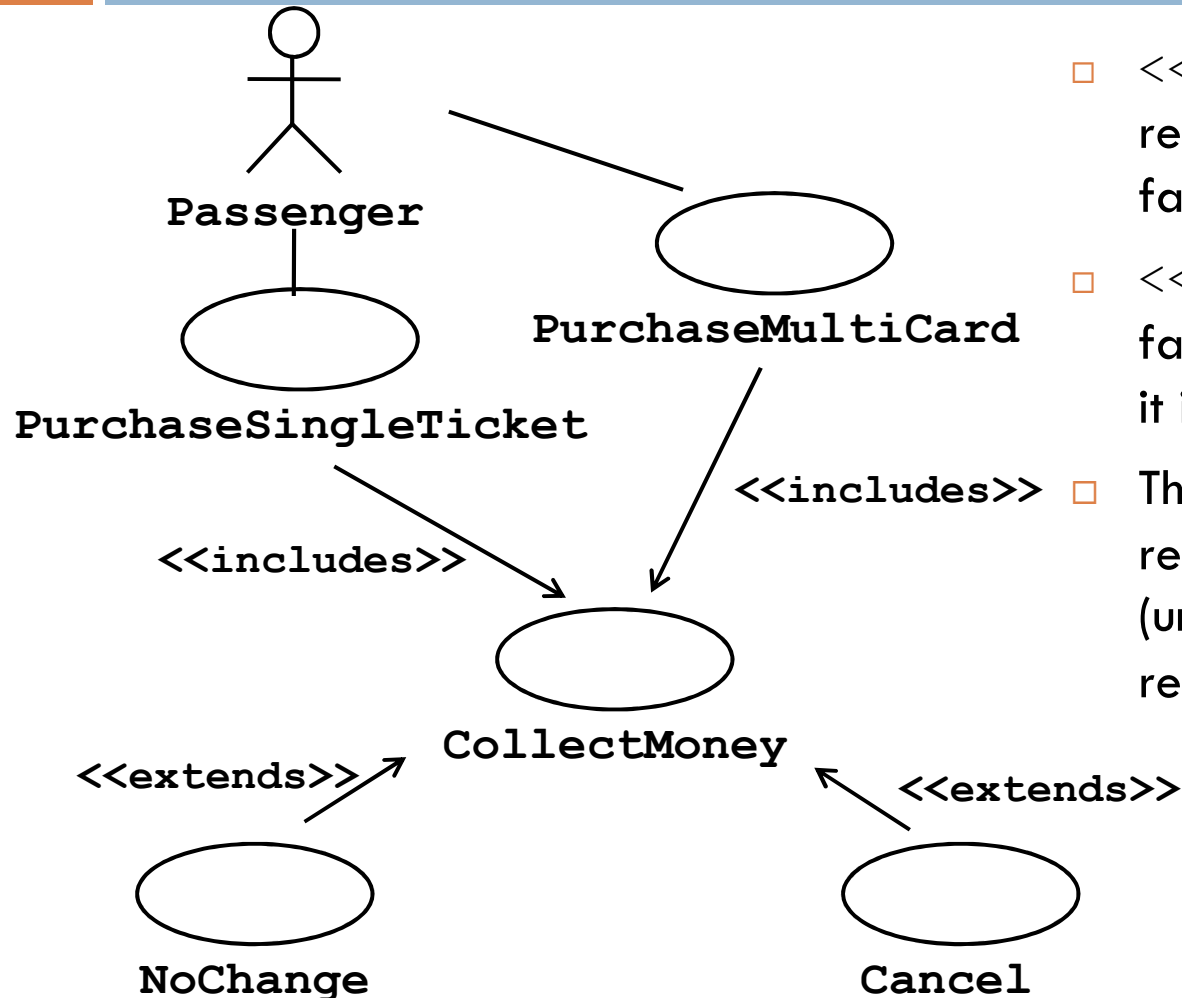
Exceptional cases!

# The <<extends>> Relationship



- <<extends>> relationships represent exceptional or seldom invoked cases.
- The exceptional event flows are factored out of the main event flow for clarity.
- Use cases representing exceptional flows can extend more than one use case.
- The direction of a <<extends>> relationship is to the extended use case

# The <<includes>> Relationship



- <<includes>> relationship represents behavior that is factored out of the use case.
- <<includes>> behavior is factored out for reuse, not because it is an exception.
- The direction of a <<includes>> relationship is to the using use case (unlike <<extends>> relationships).

# Advantages of Use Case



- Determining requirements
  - ▣ New use cases often generate new requirements as the system is analyzed and the design takes shape.
- Communicating with clients
  - ▣ Their notational simplicity makes use case diagrams a good way for developers to communicate with clients.
- Generating test cases
  - ▣ The collection of scenarios for a use case may suggest a suite of test cases for those scenarios.

# State Machine Diagram



- State machine diagram is a behavior diagram which shows discrete behavior of a part of designed system through finite state transitions. State machine diagrams can also be used to express the usage protocol of part of a system. Two kinds of state machines defined in UML are
  - ▣ Behavioral State Machine
  - ▣ Protocol State Machine



# Behavioral State Machine Diagram



- Behavioral state machine is specialization of behavior and is used to specify discrete behavior of a part of designed system through finite state transitions.
- Behavior is modeled as a traversal of a graph of state nodes connected with transitions.
- Transitions are triggered by the dispatching of series of events. During the traversal, the state machine could also execute some activities.

# Protocol State Machine Diagram

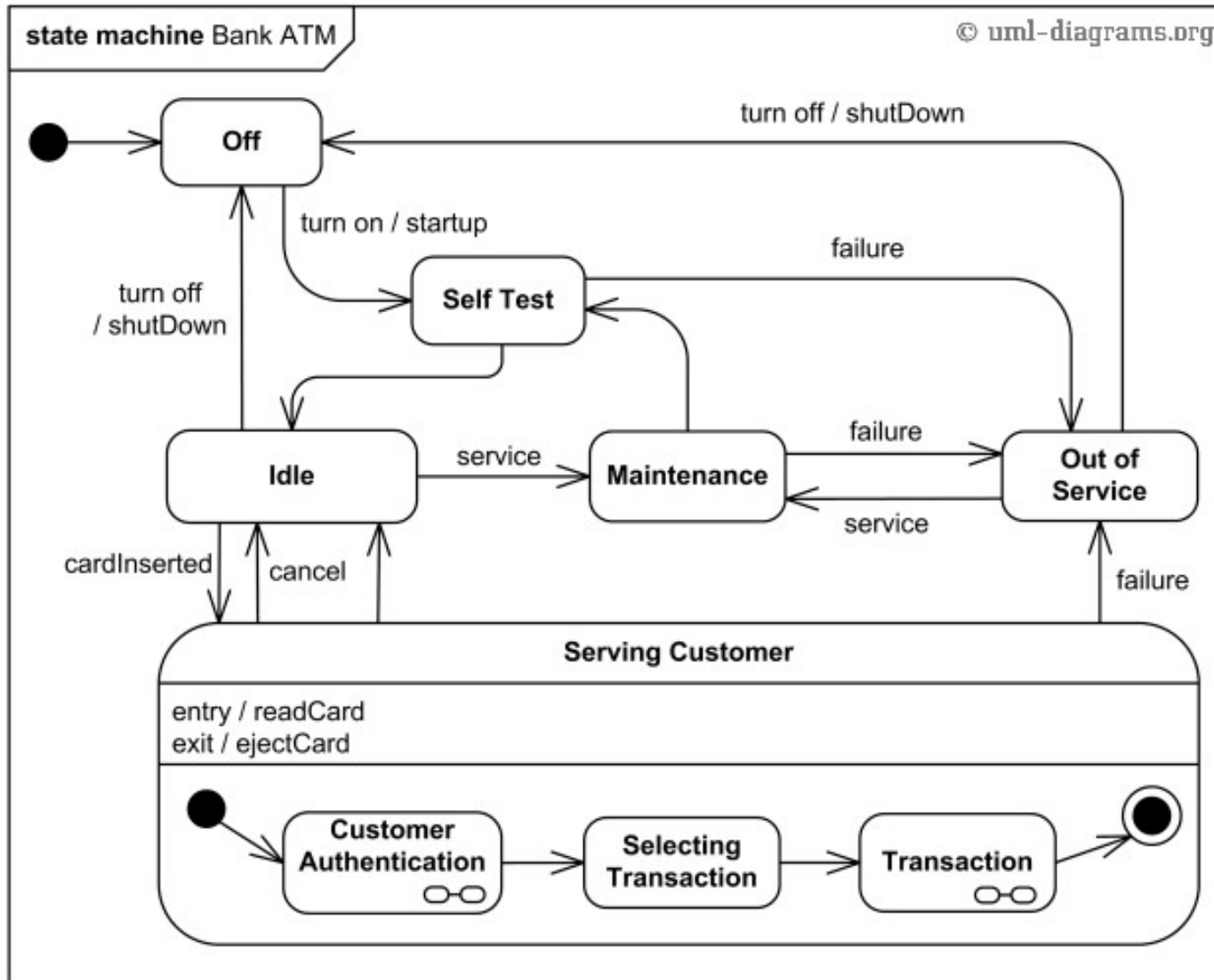
- Protocol state machine diagrams are used to express a usage protocol or a lifecycle of some classifier. It shows which operations of the classifier may be called in each state of the classifier, under which specific conditions, and satisfying some optional post-conditions after the classifier transitions to a target state.
- The notation for protocol state machine is similar to the one of behavioral state machines. The keyword **{protocol}** is placed close to the name of the state machine to differentiate protocol state machine diagrams.

# Purpose of State Machine Diagram

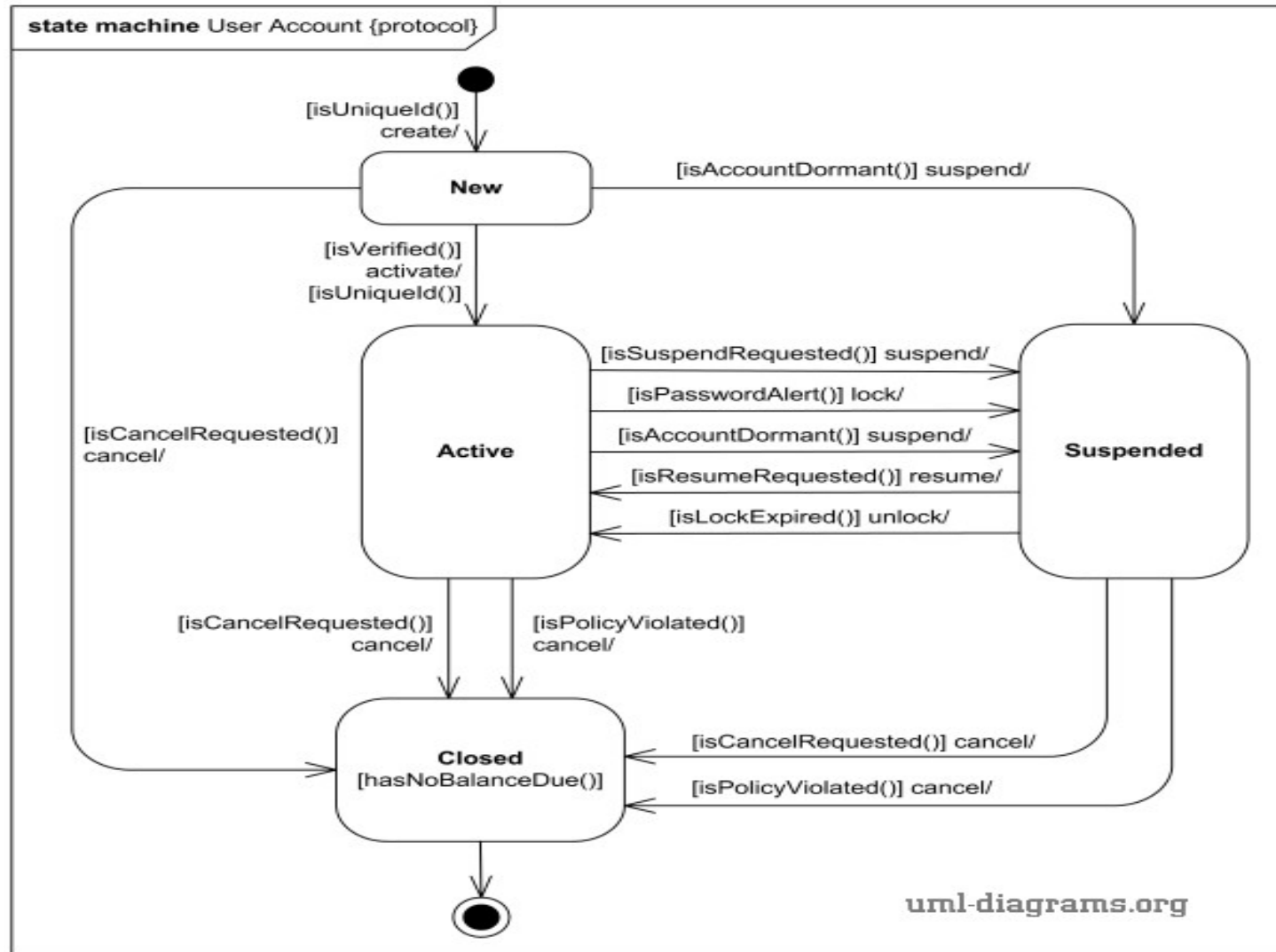


- Create a state machine diagram to
  - ▣ explore the complex behavior of a class, actor, subsystem, or component;
  - ▣ model real-time systems.

# Example State Machine Diagram 1



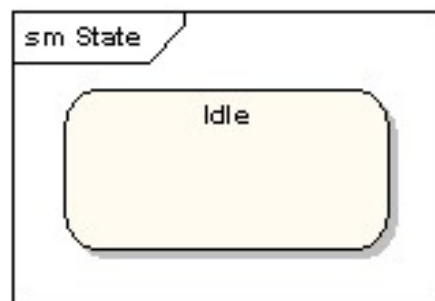
# Example State Machine Diagram 2



# State Machine Diagram Elements

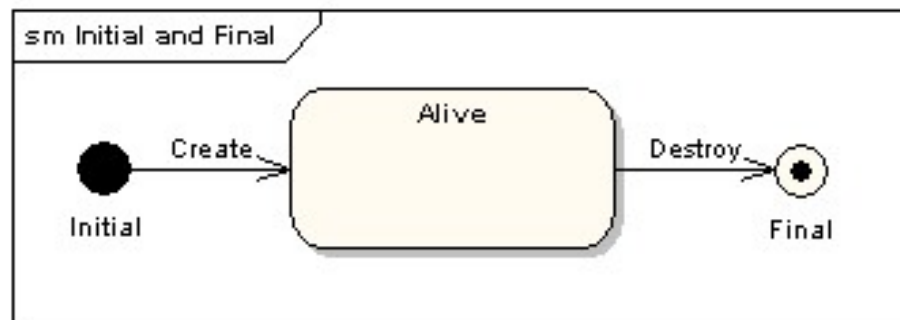
## States

A state is denoted by a round-cornered rectangle with the name of the state written inside it.



## Initial and Final States

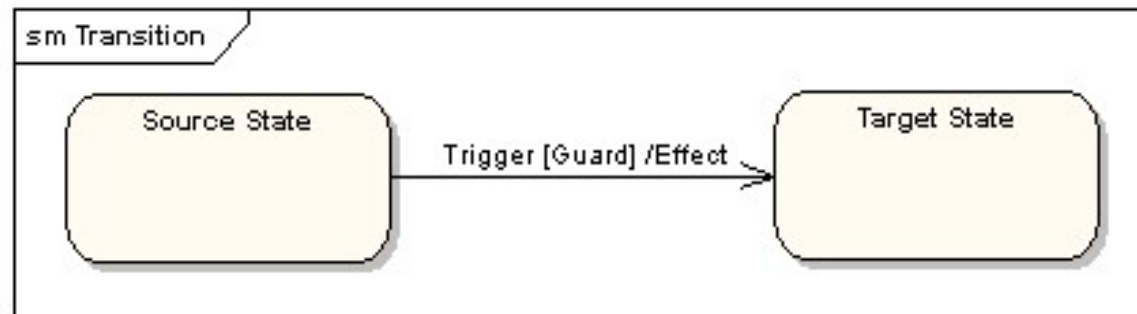
The initial state is denoted by a filled black circle and may be labeled with a name. The final state is denoted by a circle with a dot inside and may also be labeled with a name.



# State Machine Diagram Elements

## Transitions

Transitions from one state to the next are denoted by lines with arrowheads. A transition may have a trigger, a guard and an effect, as below.

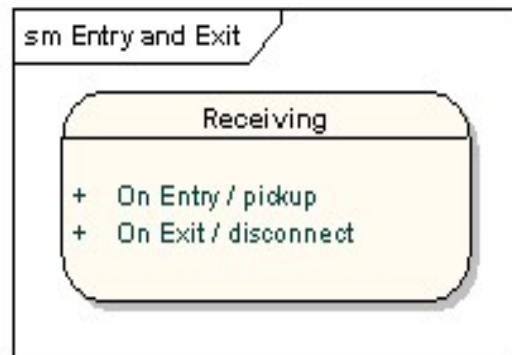


"Trigger" is the cause of the transition, which could be a signal, an event, a change in some condition, or the passage of time. "Guard" is a condition which must be true in order for the trigger to cause the transition. "Effect" is an action which will be invoked directly on the object that owns the state machine as a result of the transition.

# State Machine Diagram Elements

## State Actions

In the transition example above, an effect was associated with the transition. If the target state had many transitions arriving at it, and each transition had the same effect associated with it, it would be better to associate the effect with the target state rather than the transitions? This can be done by defining an entry action for the state. The diagram below shows a state with an entry action and an exit action.



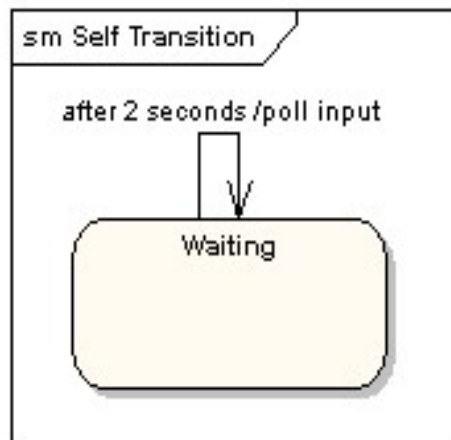
It is also possible to define actions that occur on events, or actions that always occur. It is possible to define any number of actions of each type.



# State Machine Diagram Elements

## Self-Transitions

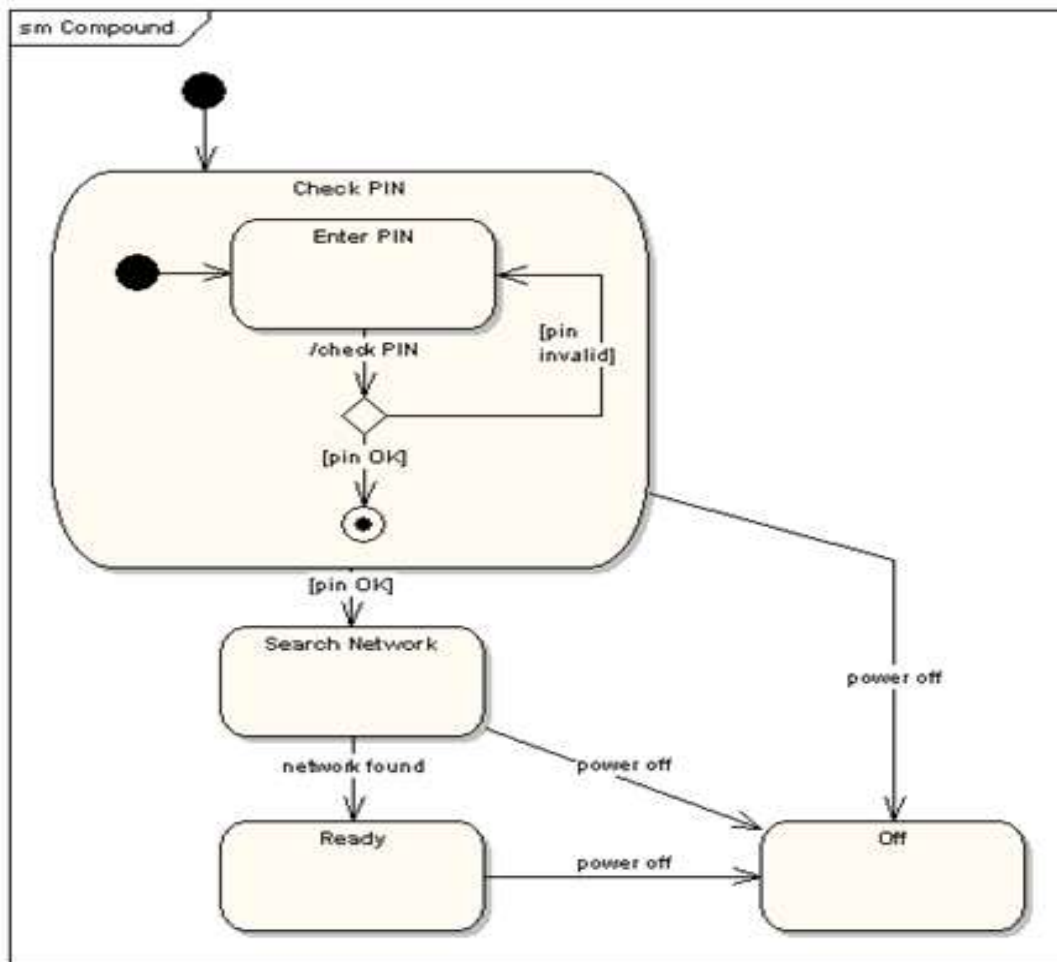
A state can have a transition that returns to itself, as in the following diagram. This is most useful when an effect is associated with the transition.



# State Machine Diagram Elements

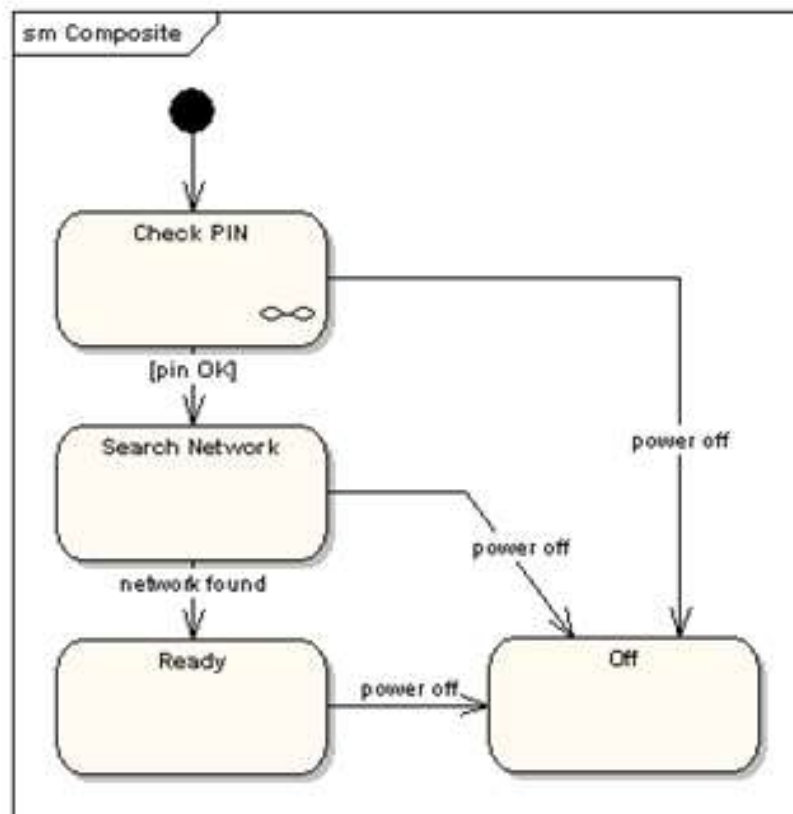
## Compound States

State machine diagram may include sub-machine diagrams, as in the example below.



# State Machine Diagram Elements

The alternative way to show the same information is as follows.

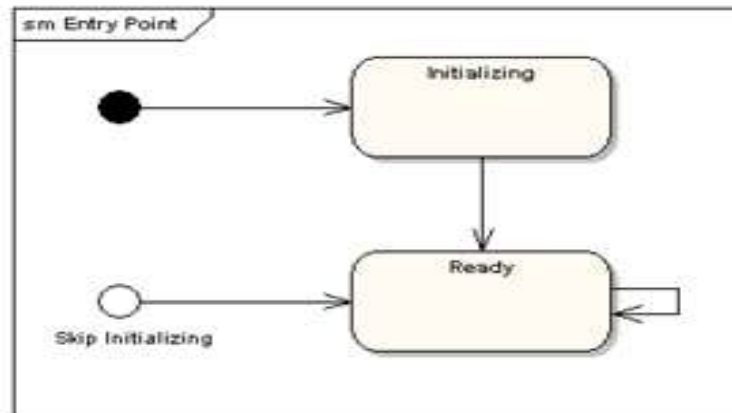


The notation in the above version indicates that the details of the Check PIN sub-machine are shown in a separate diagram.

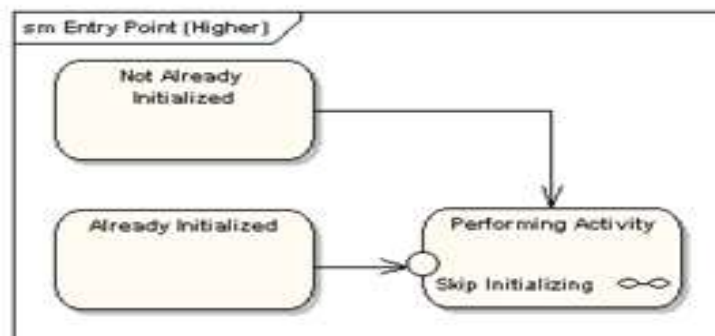
# State Machine Diagram Elements

## Entry Point

Sometimes you won't want to enter a sub-machine at the normal initial state. For example, in the following sub-machine it would be normal to begin in the "Initializing" state, but if for some reason it wasn't necessary to perform the initialization, it would be possible to begin in the "Ready" state by transitioning to the named entry point.



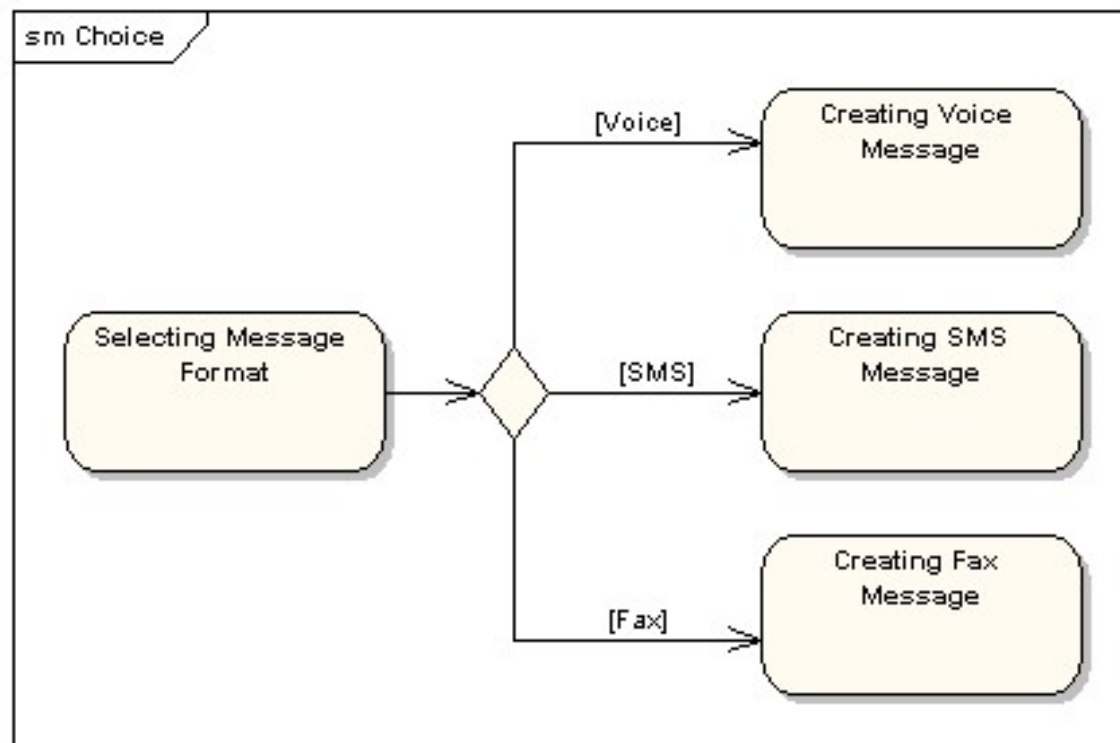
The following diagram shows the state machine one level up.



# State Machine Diagram Elements

## Choice Pseudo-State

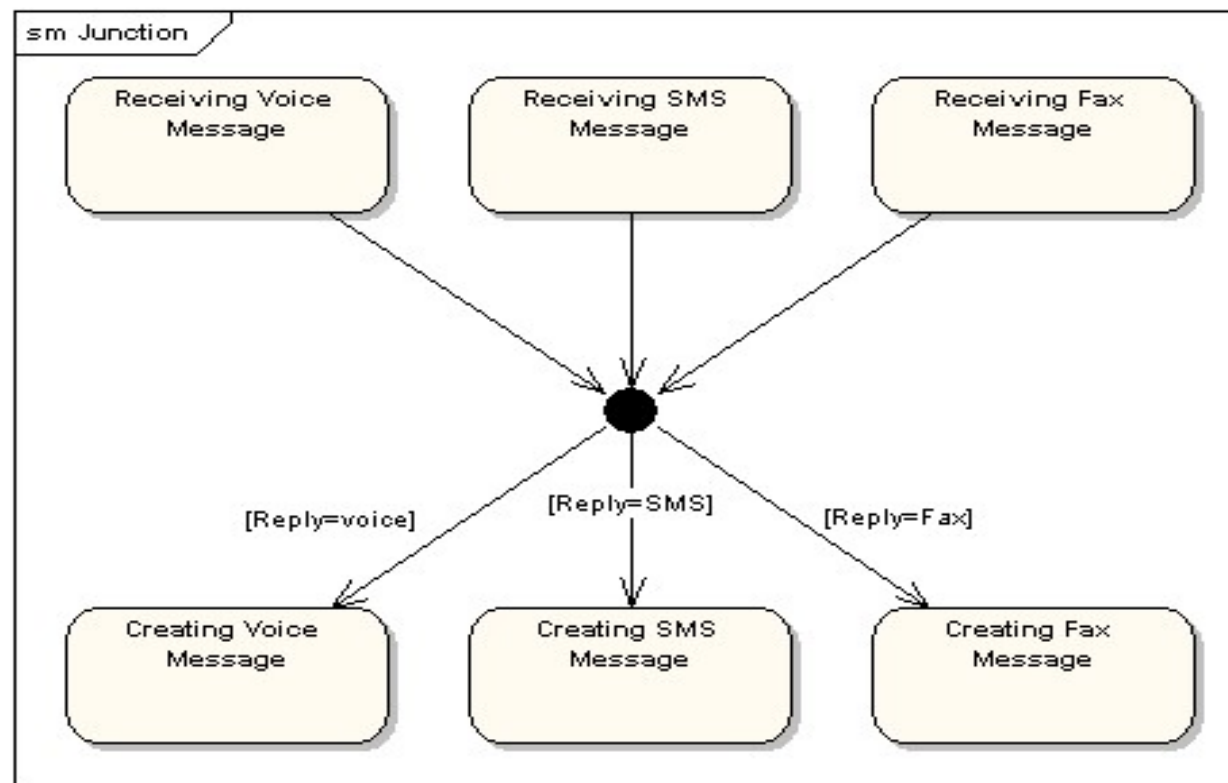
A choice pseudo-state is shown as a diamond with one transition arriving and two or more transitions leaving. The following diagram shows that whichever state is arrived at, after the choice pseudo-state, is dependent on the message format selected during execution of the previous state.



# State Machine Diagram Elements

## Junction Pseudo-State

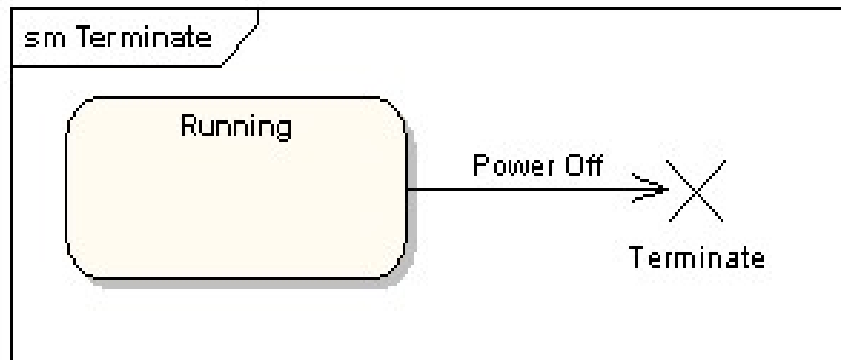
Junction pseudo-states are used to chain together multiple transitions. A single junction can have one or more incoming, and one or more outgoing, transitions; a guard can be applied to each transition. Junctions are semantic-free. A junction who splits an incoming transition into multiple outgoing transitions realizes a static conditional branch, as opposed to a choice pseudo-state which realizes a dynamic conditional branch.



# State Machine Diagram Elements

## Terminate Pseudo-State

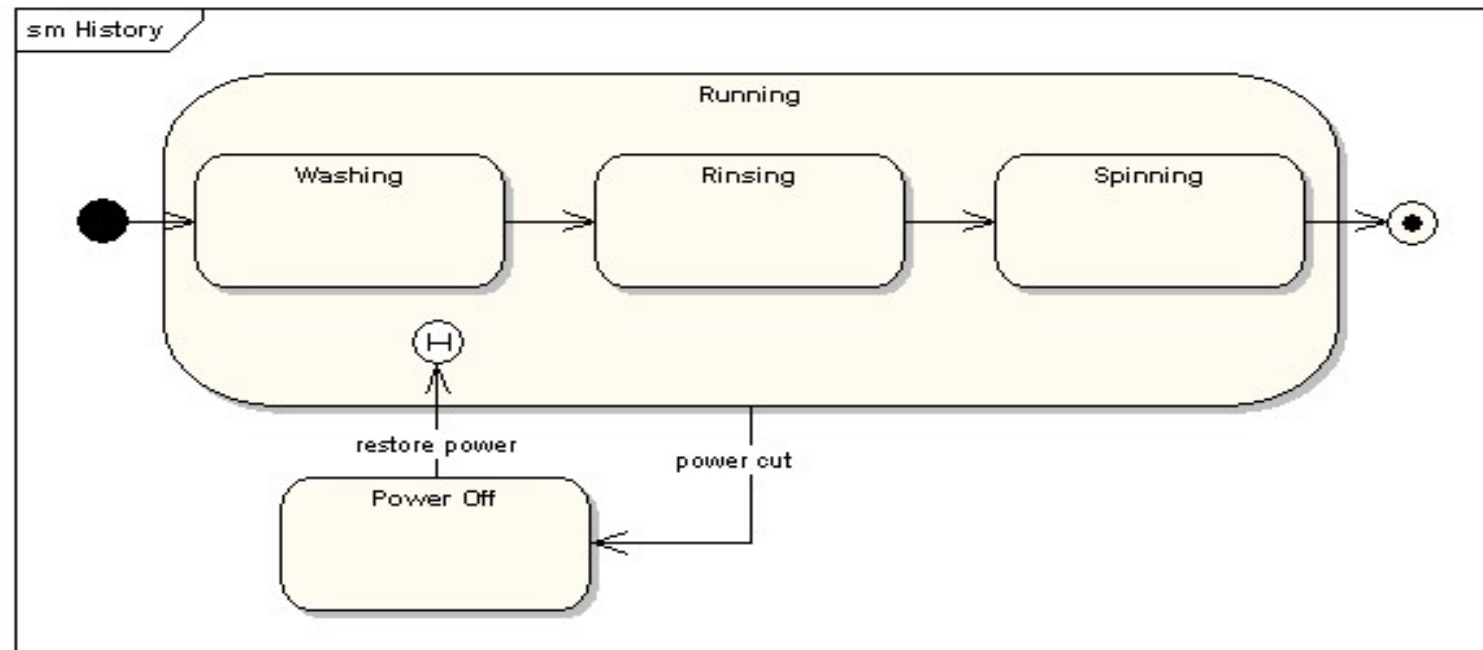
Entering a terminate pseudo-state indicates that the lifeline of the state machine has ended. A terminate pseudo-state is notated as a cross.



# State Machine Diagram Elements

## History States

A history state is used to remember the previous state of a state machine when it was interrupted. The following diagram illustrates the use of history states. The example is a state machine belonging to a washing machine.



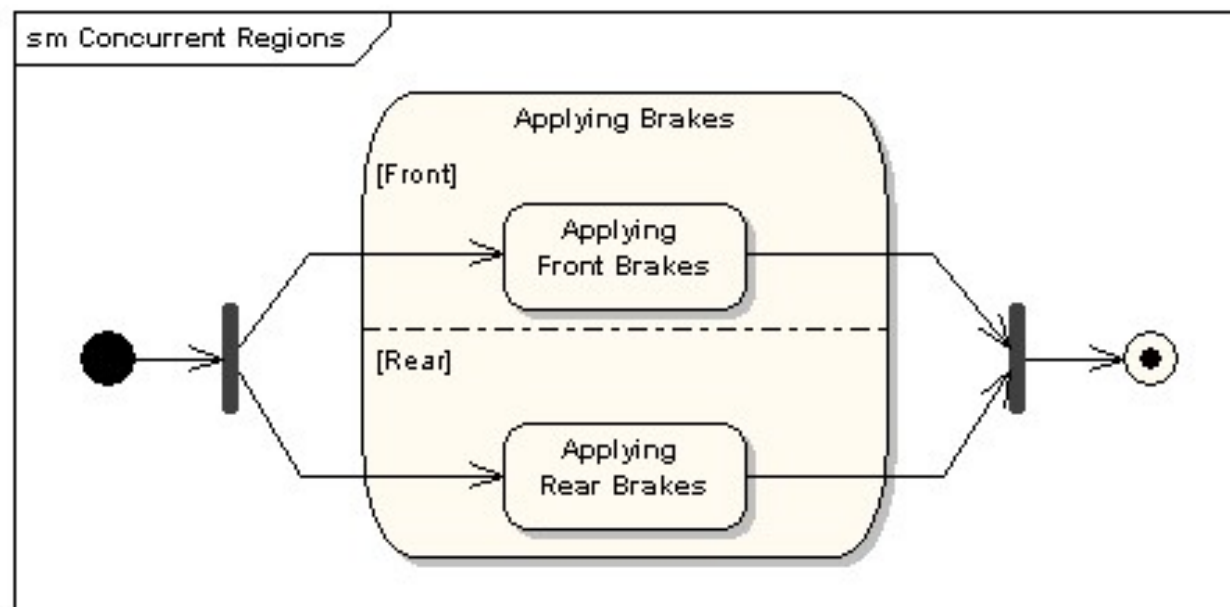
In this state machine, when a washing machine is running, it will progress from "Washing" through "Rinsing" to "Spinning". If there is a power cut, the washing machine will stop running and will go to the "Power Off" state. Then when the power is restored, the Running state is entered at the "History State" symbol meaning that it should resume where it last left-off.



# State Machine Diagram Elements

## Concurrent Regions

A state may be divided into regions containing sub-states that exist and execute concurrently. The example below shows that within the state "Applying Brakes", the front and rear brakes will be operating simultaneously and independently. Notice the use of fork and join pseudo-states, rather than choice and merge pseudo-states. These symbols are used to synchronize the concurrent threads.



# Timing Diagrams

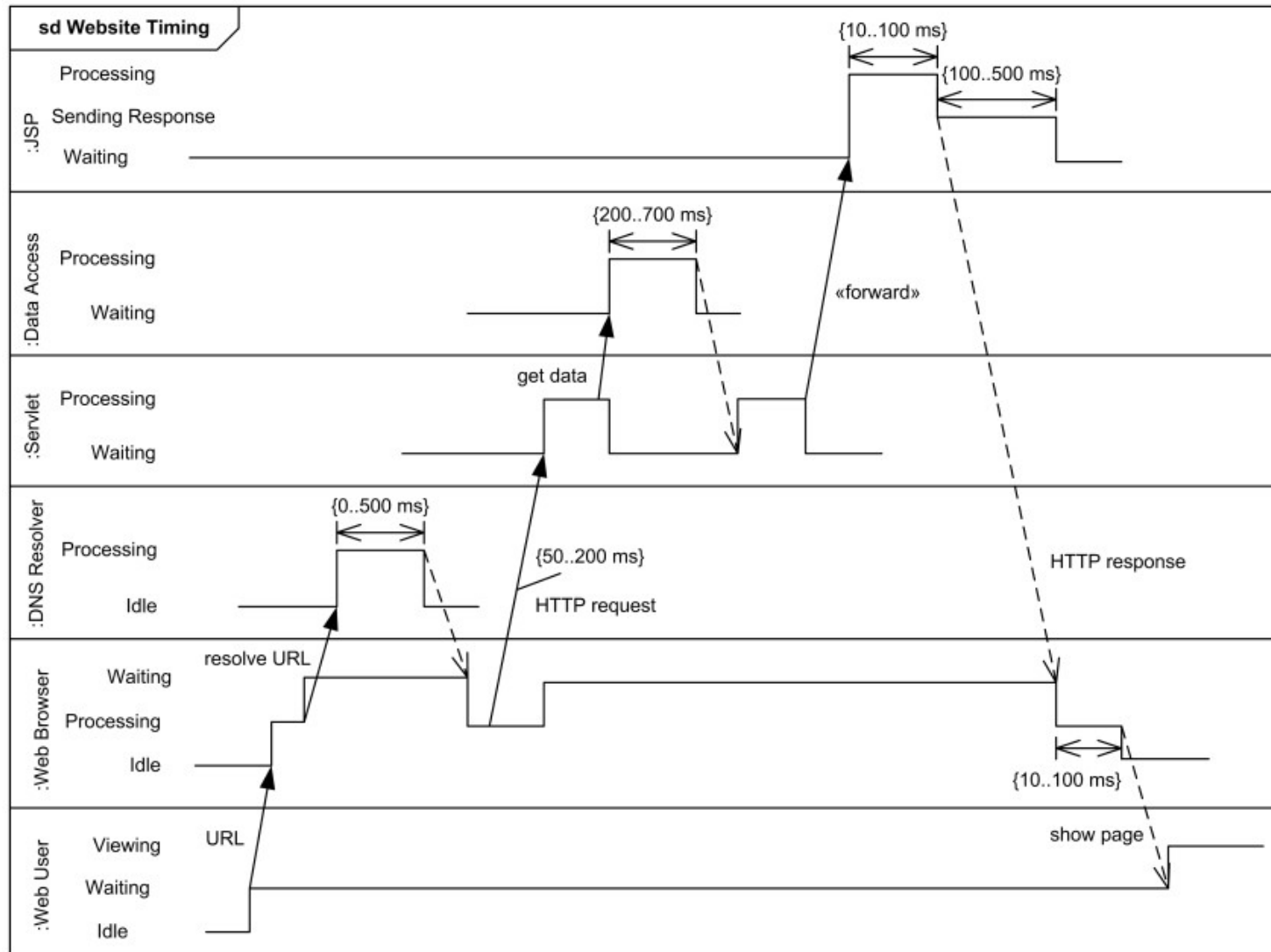
- Timing diagrams are interaction diagrams used to show interactions when a primary purpose of the diagram is to reason about time.
- Timing diagrams focus on conditions changing within and among lifelines along a linear time axis.
- Timing diagrams describe behavior of both individual classifiers and interactions of classifiers, focusing attention on time of events causing changes in the modeled conditions of the lifelines.

# Purpose of Timing Diagram



- Timing diagrams are often used in the design of embedded software, such as control software for a fuel injection system in an automobile, although they occasionally have their uses for business software too.
- Timing diagram should be created when timing, not sequence, is of critical importance.

# Example Timing Diagram 1



# Sequence Diagram



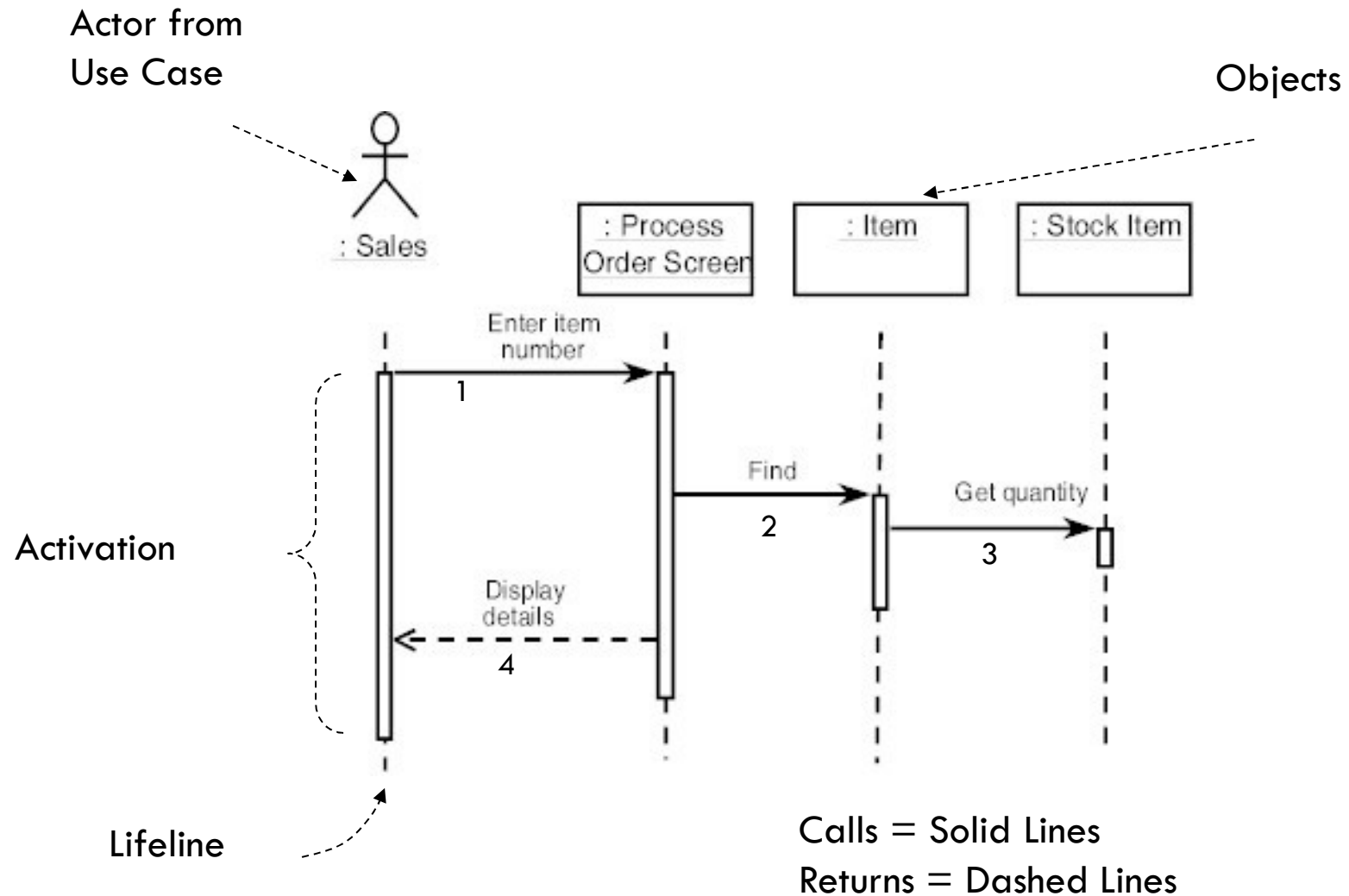
- A sequence diagram is a form of interaction diagram which shows objects as lifelines running down the page, with their interactions over time represented as messages drawn as arrows from the source lifeline to the target lifeline. Sequence diagrams are good at showing which objects communicate with which other objects; and what messages trigger those communications. Sequence diagrams are not intended for showing complex procedural logic.

# Purpose of Sequence Diagram

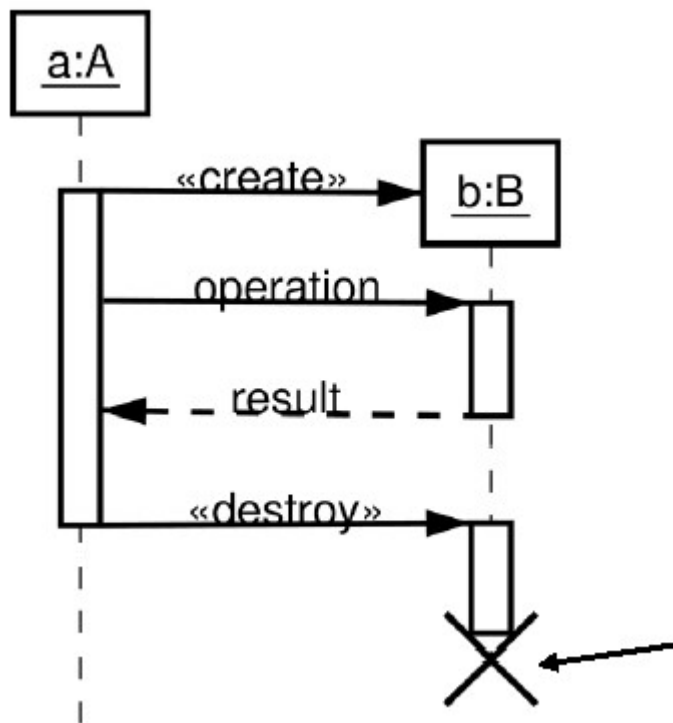


- Depict work flow, message passing and how elements in general cooperate over time to achieve a result
- Capture the flow of information and responsibility throughout the system, early in analysis; messages between elements eventually become method calls in the Class model
- Make explanatory models for Use Case scenarios; by creating a Sequence diagram with an Actor and elements involved in the Use Case, you can model the sequence of steps the user and the system undertake to complete the required tasks

# Sequence Diagram Format



# Sequence Diagram : Destruction

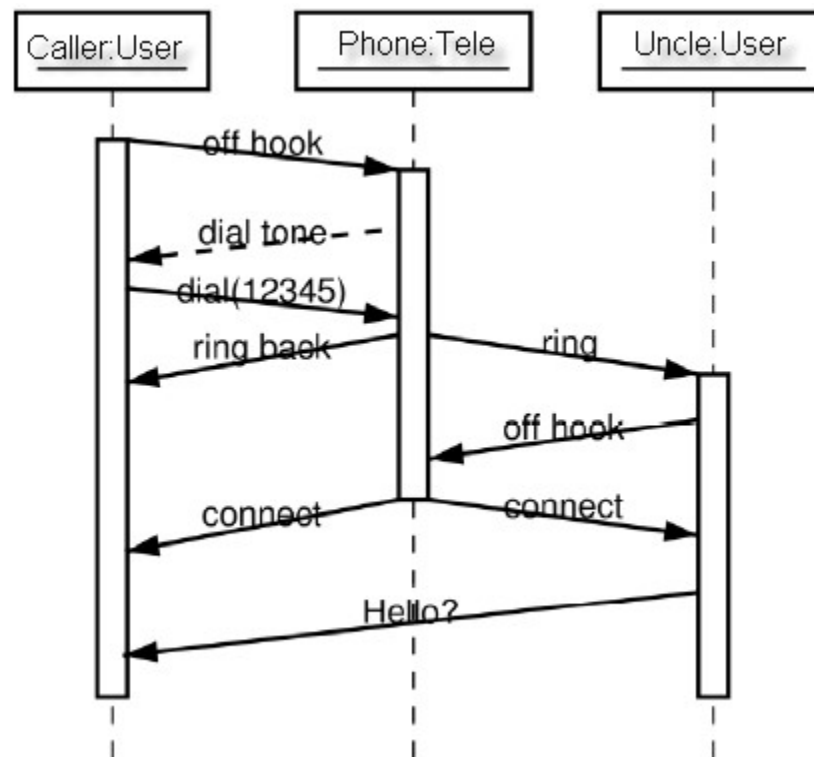


Shows Destruction of b  
(and Construction)



# Sequence Diagram : Timing

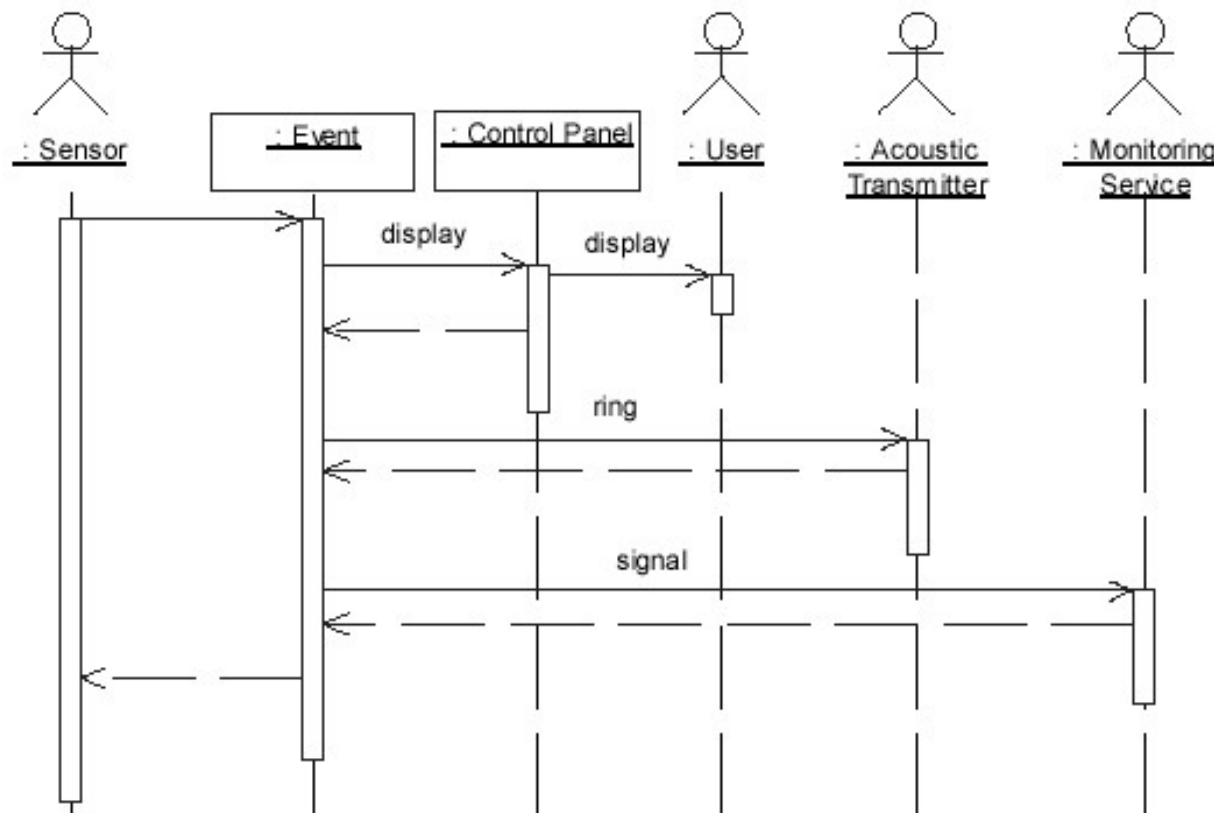
Slanted Lines show propagation delay of messages  
Good for modeling real-time systems



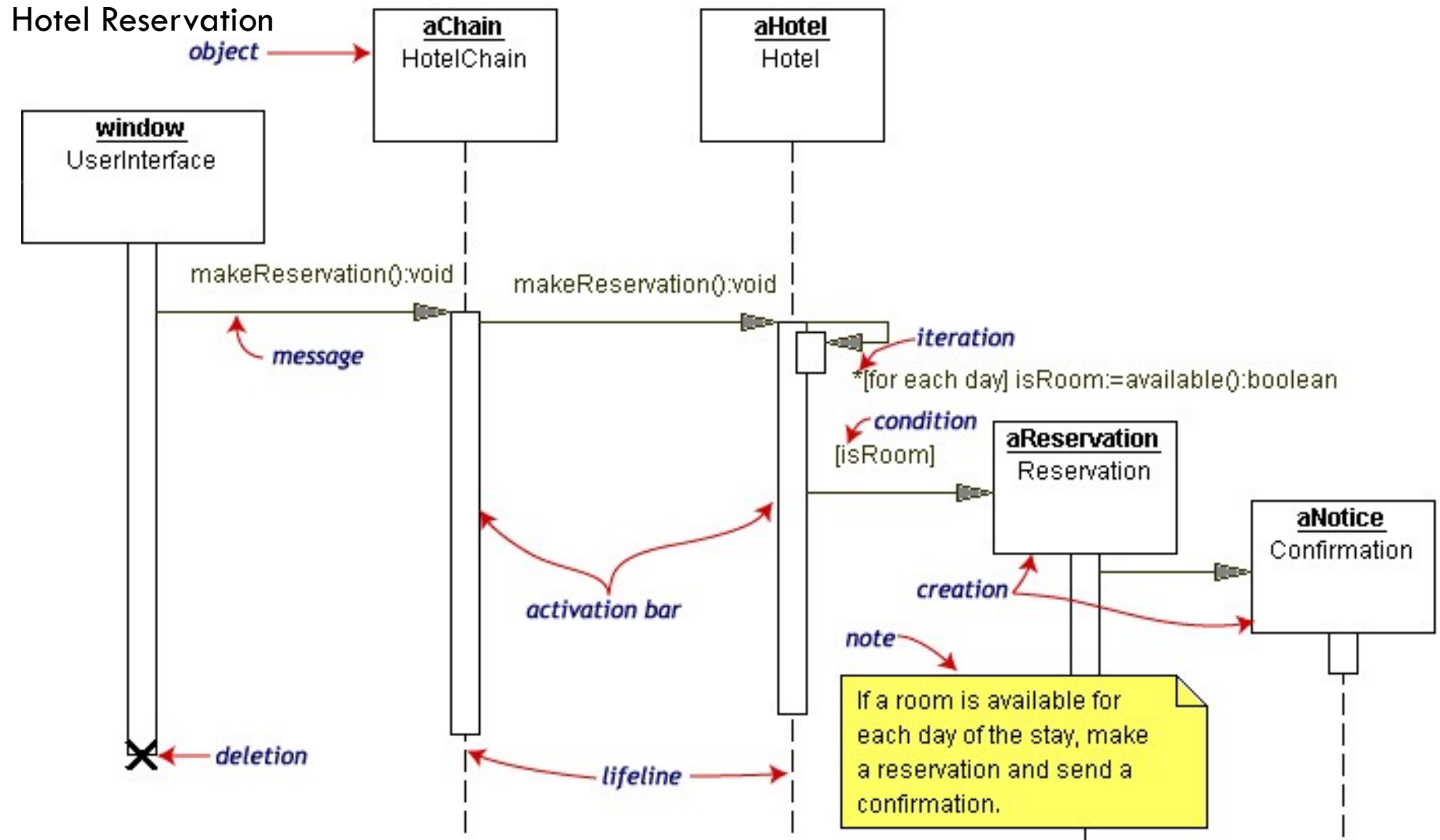
If messages cross this is usually problematic – race conditions

# Sequence Diagram Example 1

- When the alarm goes off, it rings the alarm, puts a message on the display, notifies the monitoring service



# Sequence Diagram Example 2



# Communication Diagram



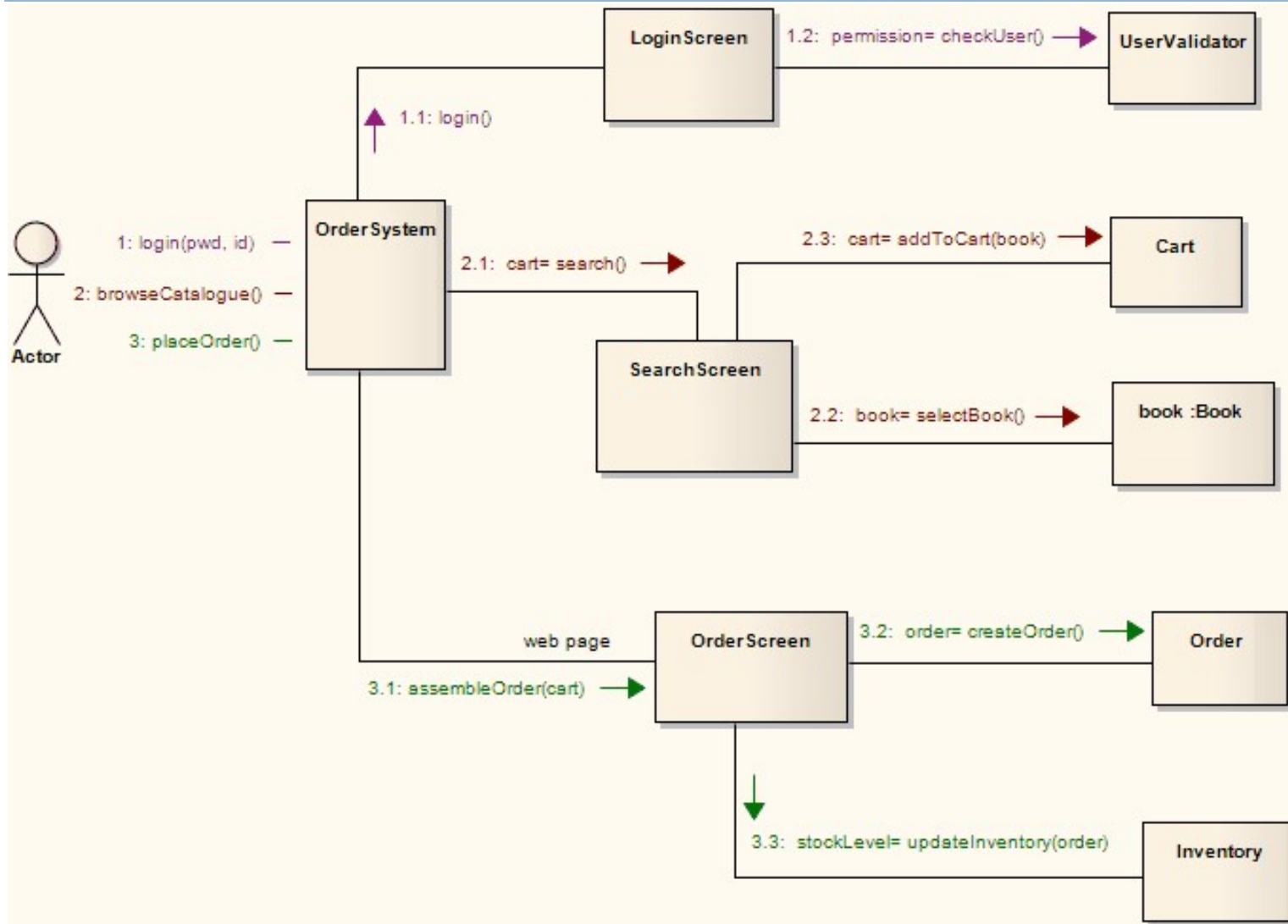
- Communication diagrams, formerly known as collaboration diagrams, are used to explore the dynamic nature of software. Communication diagrams show the message flow between objects in an object-oriented application, and also imply the basic associations (relationships) between classes.

# Purpose of Communication Diagram

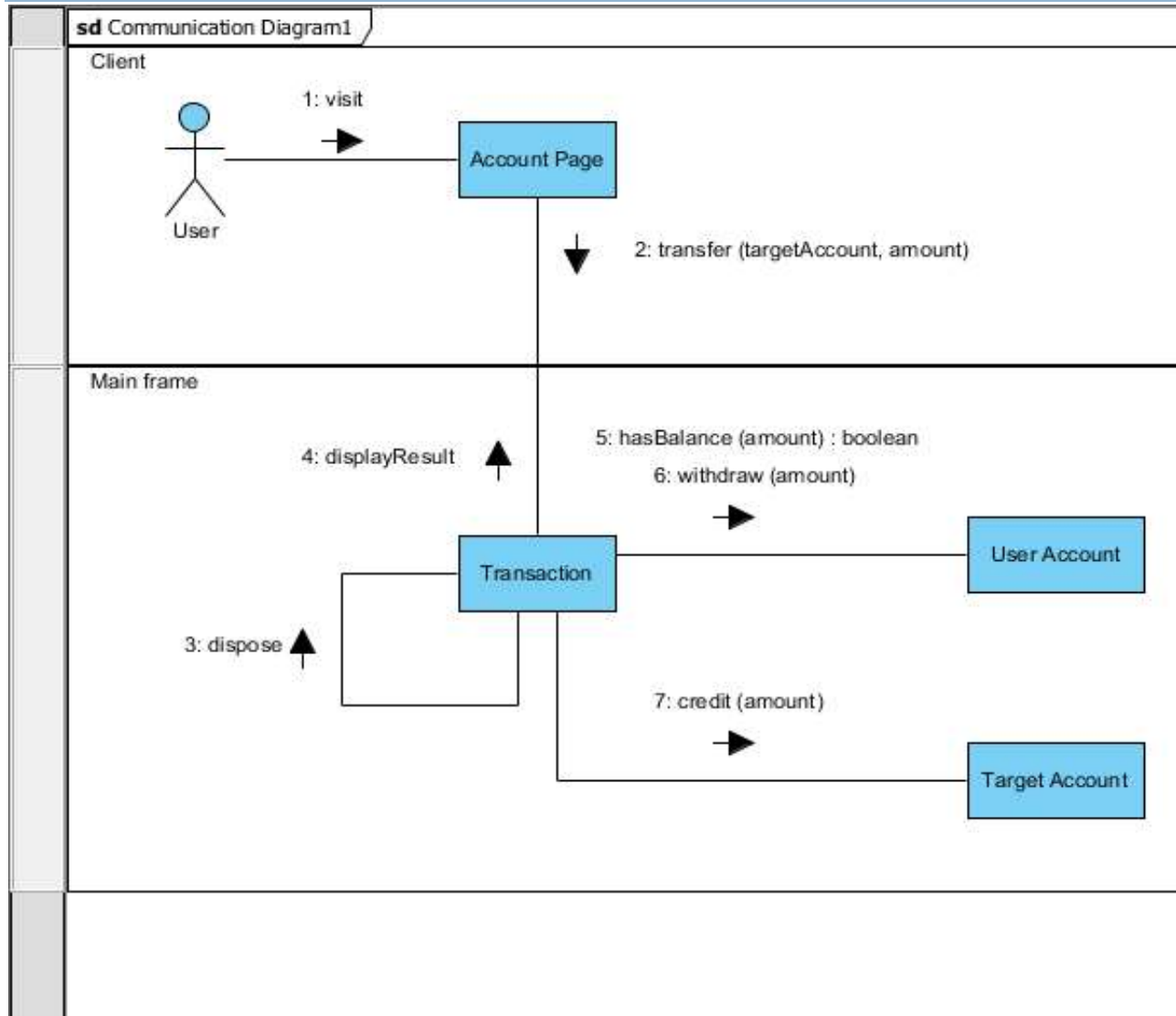


- Provide a bird's-eye view of a collection of collaborating objects, particularly within a real-time environment
- Provide an alternate view to sequence diagrams
- Allocate functionality to classes by exploring the behavioral aspects of a system
- Model the logic of the implementation of a complex operation, particularly one that interacts with a large number of other objects
- Explore the roles that objects take within a system, as well as the different relationships in which they are involved when in those roles

# Communication Diagram Example 1



# Communication Diagram Example 2



# Interaction Overview Diagram

- Interaction overview diagrams are new to UML 2. They are a variant of an activity diagram where each bubble on the diagram represents another interaction diagram.

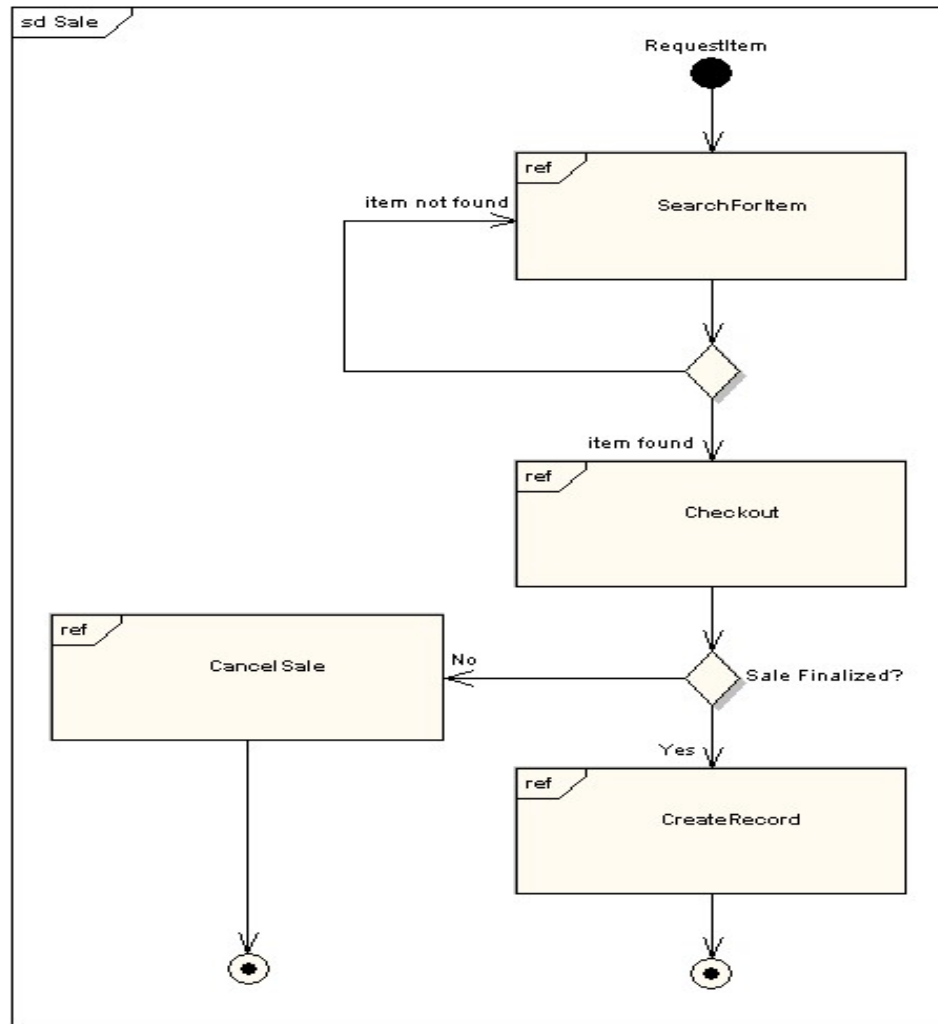


# Purpose of Interaction Overview



- Overview the flow of control within a business process
- Overview the detailed logic of a software process
- Connect several diagrams together

# Interaction Overview Example

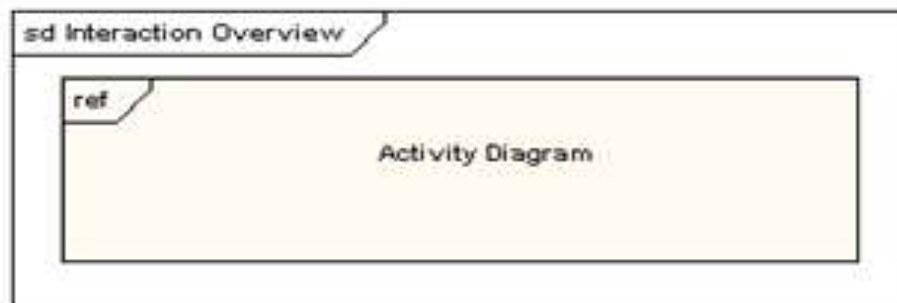


# Elements of Interaction Overview

Most of the notation for interaction overview diagrams is the same for activity diagrams. For example, initial, final, decision, merge, fork and join nodes are all the same. However, interaction overview diagrams introduce two new elements: interaction occurrences and interaction elements.

## Interaction Occurrence

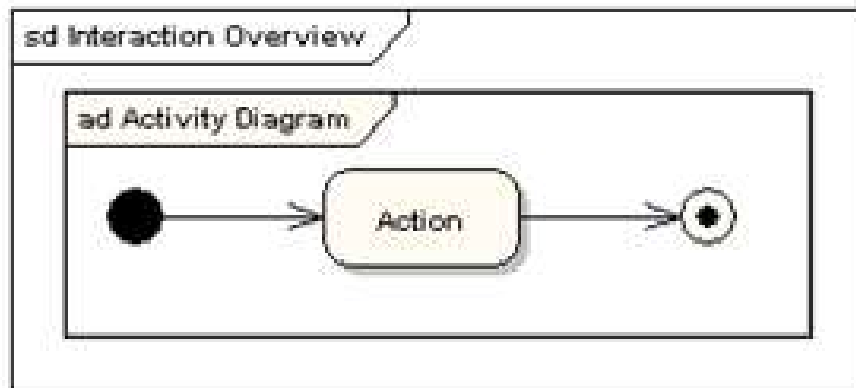
Interaction occurrences are references to existing interaction diagrams. An interaction occurrence is shown as a reference frame; that is, a frame with "ref" in the top-left corner. The name of the diagram being referenced is shown in the center of the frame.



# Elements of Interaction Overview

## Interaction Element

Interaction elements are similar to interaction occurrences, in that they display a representation of existing interaction diagrams within a rectangular frame. They differ in that they display the contents of the referenced diagram inline.



END OF BEHAVIORAL DIAGRAMS

# Structural Diagrams



- Class Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram

# Class Diagrams

- Gives an overview of a system by showing its classes and the relationships among them.
  - ▣ Class diagrams are static
  - ▣ they display what interacts but not what happens when they do interact
- Also shows attributes and operations of each class
- Good way to describe the overall architecture of system components

# Purpose of Class Diagram



- To illustrate relationships between Classes and Interfaces; Generalizations, Aggregations and Associations are all valuable in reflecting inheritance, composition or usage, and connections, respectively

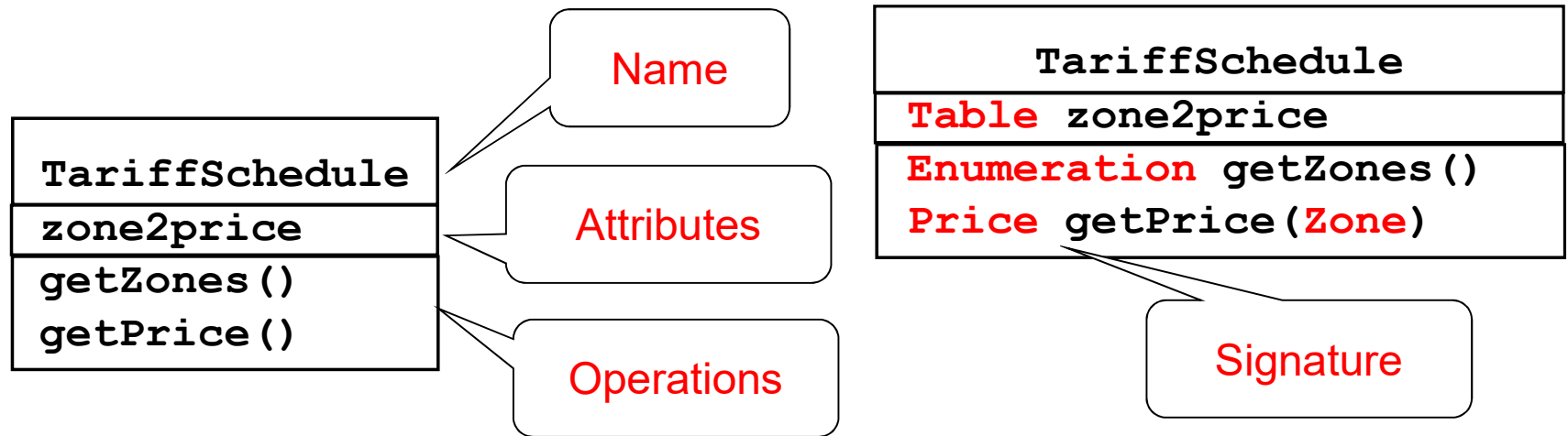


# Class Diagram Perspectives



- We draw Class Diagrams under three perspectives
  - ▣ Conceptual
    - Software independent
    - Language independent
  - ▣ Specification
    - Focus on the interfaces of the software
  - ▣ Implementation
    - Focus on the implementation of the software

# Classes – Not Just for Code



- A **class** represent a concept
- A class encapsulates state (**attributes**) and behavior (**operations**).
- Each attribute has a **type**.
- Each operation has a **signature**.
- The class name is the only mandatory information.

# UML Class Notation

- A class is a rectangle divided into three parts
  - ▣ Class name
  - ▣ Class attributes (i.e. data members, variables)
  - ▣ Class operations (i.e. methods)
- Modifiers
  - ▣ Private: -
  - ▣ Public: +
  - ▣ Protected: #
  - ▣ Static: Underlined (i.e. shared among all members of the class)
- Abstract class: Name in italics

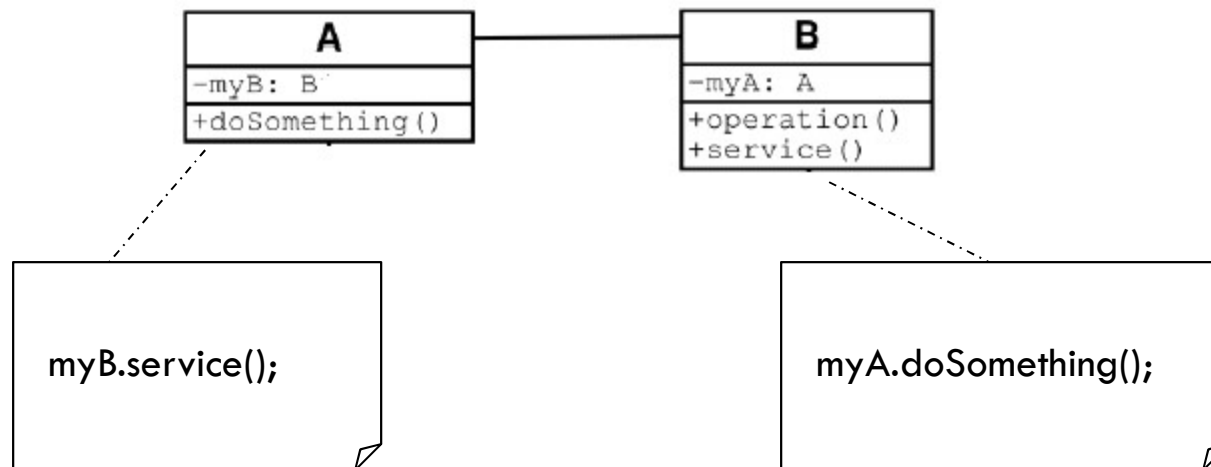
| Employee   |
|--|
| -Name : string<br>+ID : long<br>#Salary : double                                     |
| +getName() : string<br>+setName()<br>-calcInternalStuff(in x : byte, in y : decimal) |

# UML Class Notation

- Lines or arrows between classes indicate relationships
  - ▣ Association
    - A relationship between instances of two classes, where one class must know about the other to do its work, e.g. client communicates to server
    - indicated by a straight line or arrow
  - ▣ Aggregation
    - An association where one class belongs to a collection, e.g. instructor part of Faculty
    - Indicated by an empty diamond on the side of the collection
  - ▣ Composition
    - Strong form of Aggregation
    - Lifetime control; components cannot exist without the aggregate
    - Indicated by a solid diamond on the side of the collection
  - ▣ Inheritance
    - An inheritance link indicating one class a superclass relationship, e.g. bird is part of mammal
    - Indicated by triangle pointing to superclass

# Binary Association

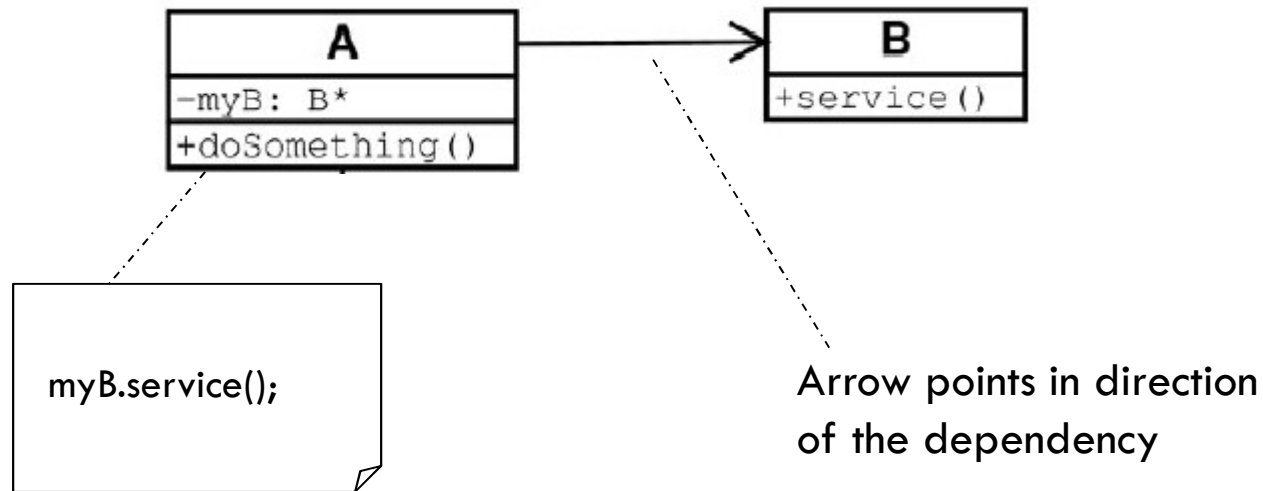
Binary Association: Both entities “Know About” each other



Optionally, may create an Associate Class

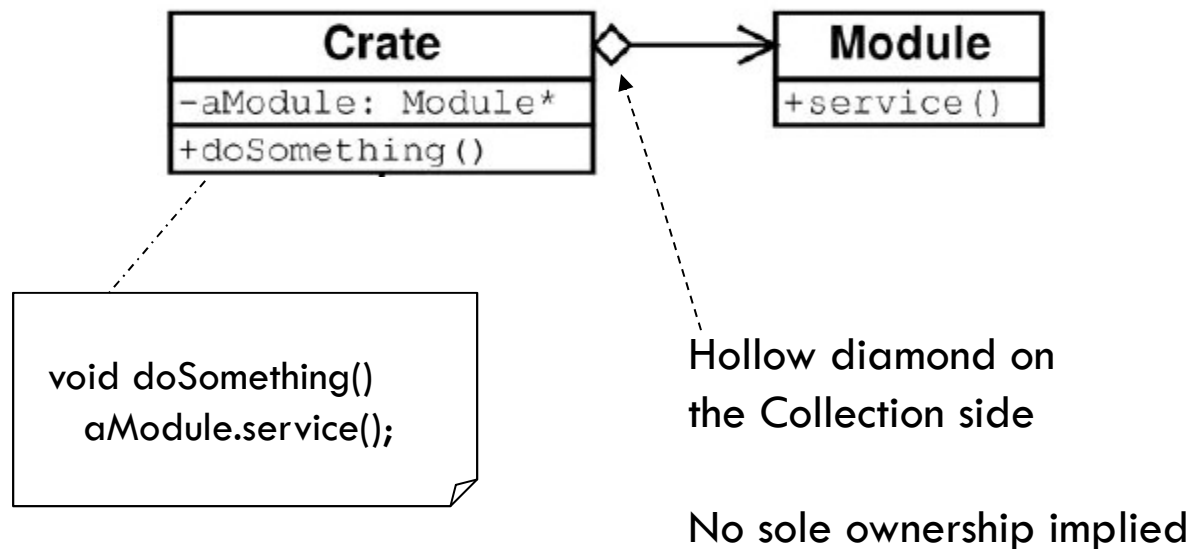
# Unary Association

A knows about B, but B knows nothing about A



# Aggregation

Aggregation is an association with a “collection-member” relationship

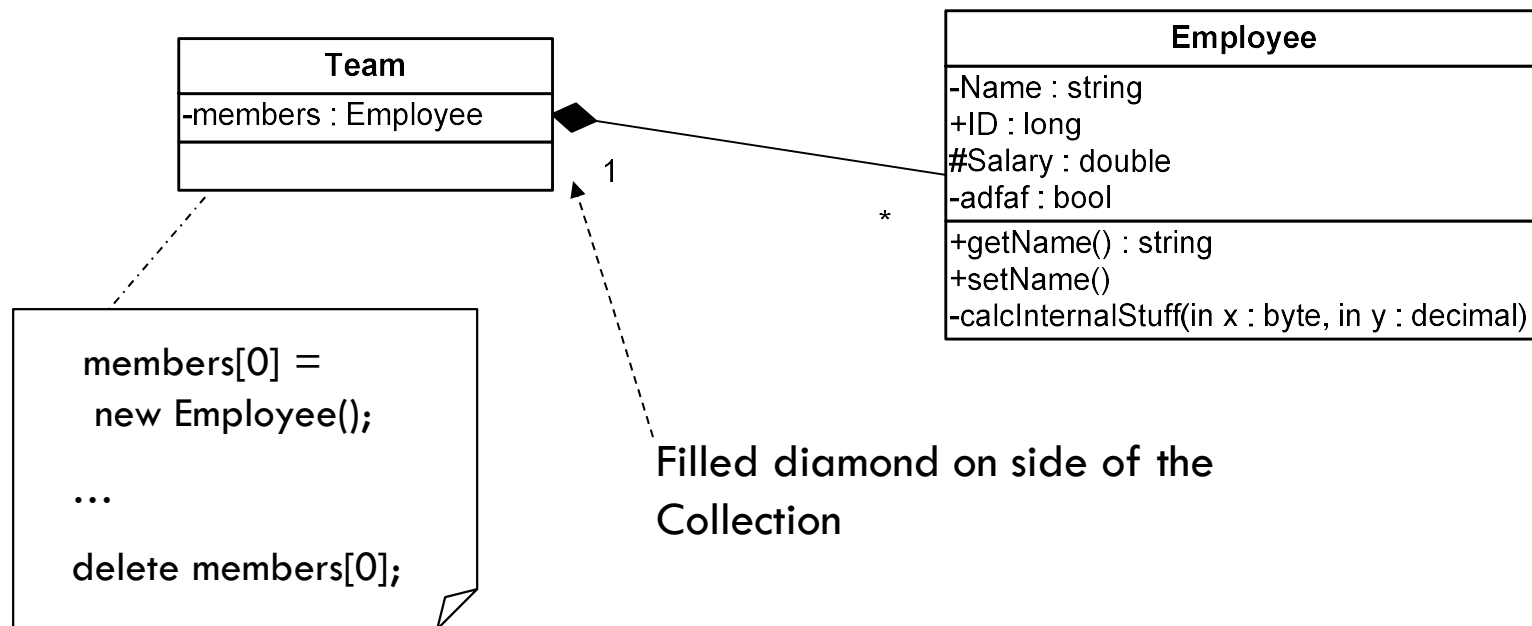


# Composition

Composition is Aggregation with:

Lifetime Control (owner controls construction, destruction)

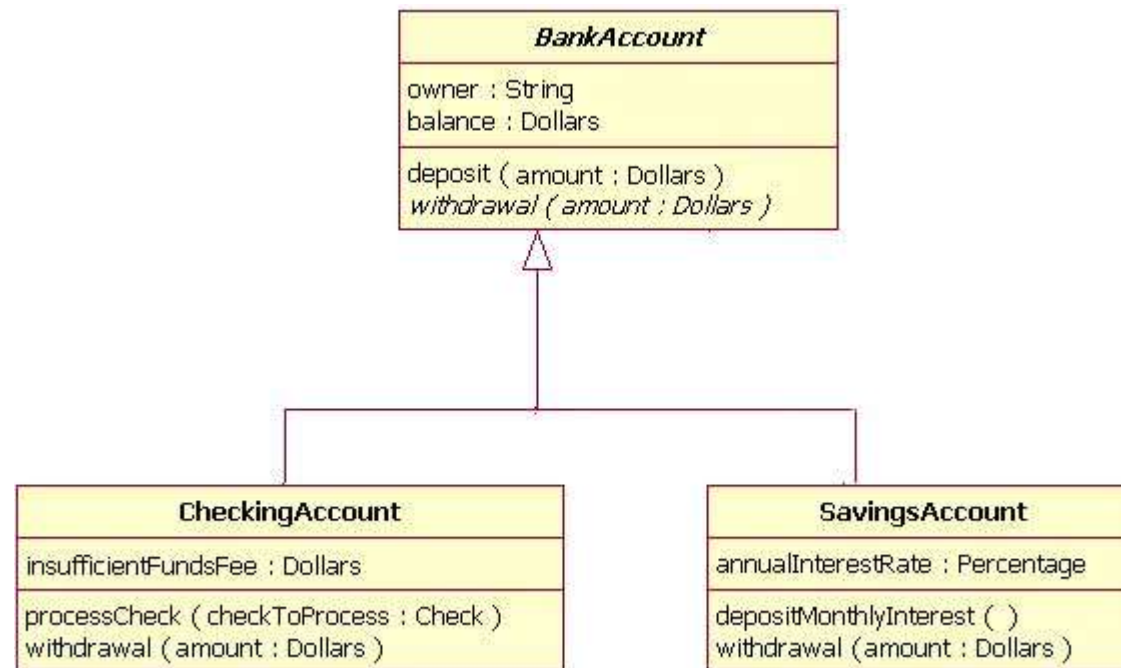
Part object may belong to only one whole object





# Inheritance

Standard concept of inheritance

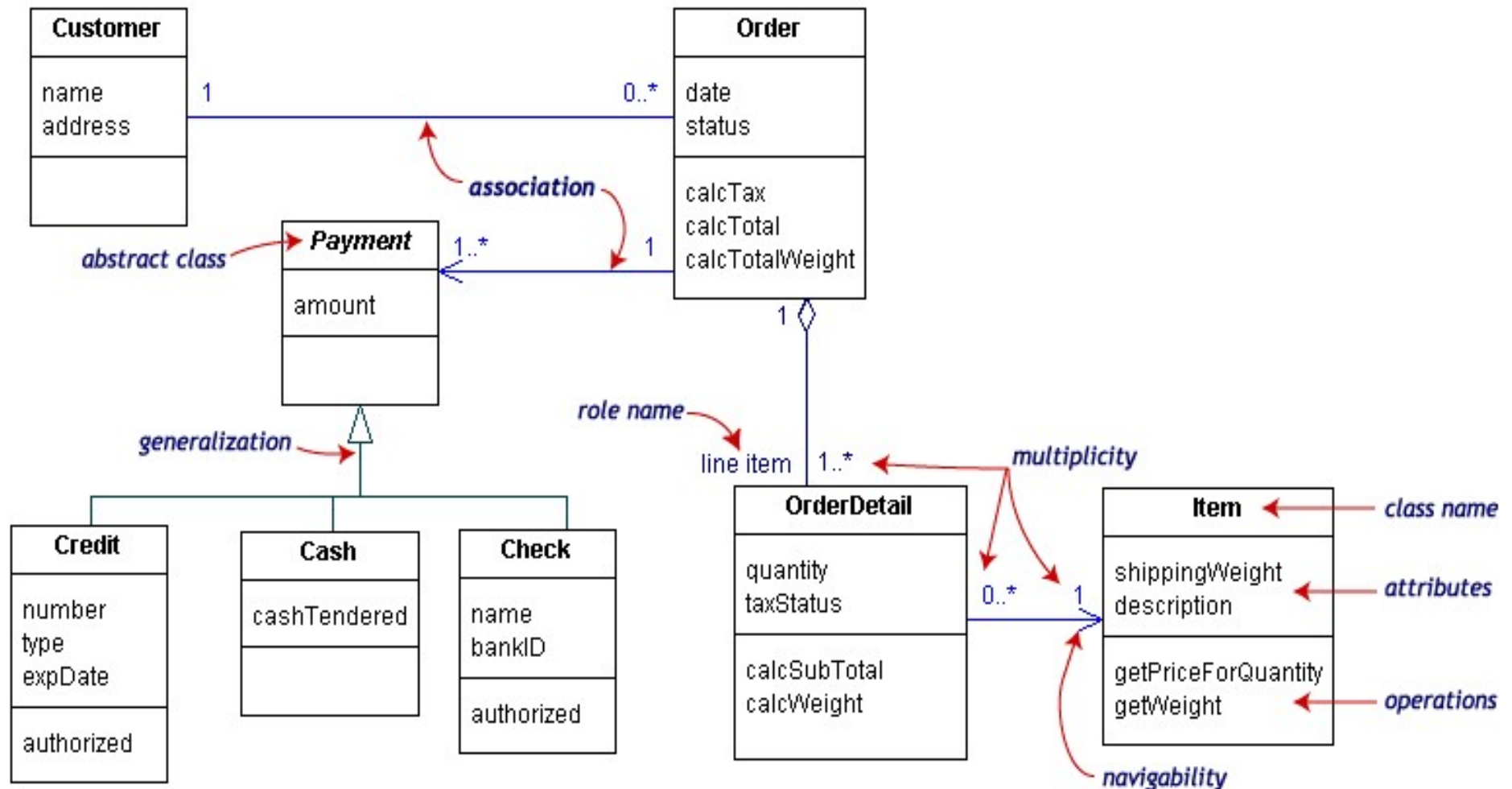


# UML Multiplicities

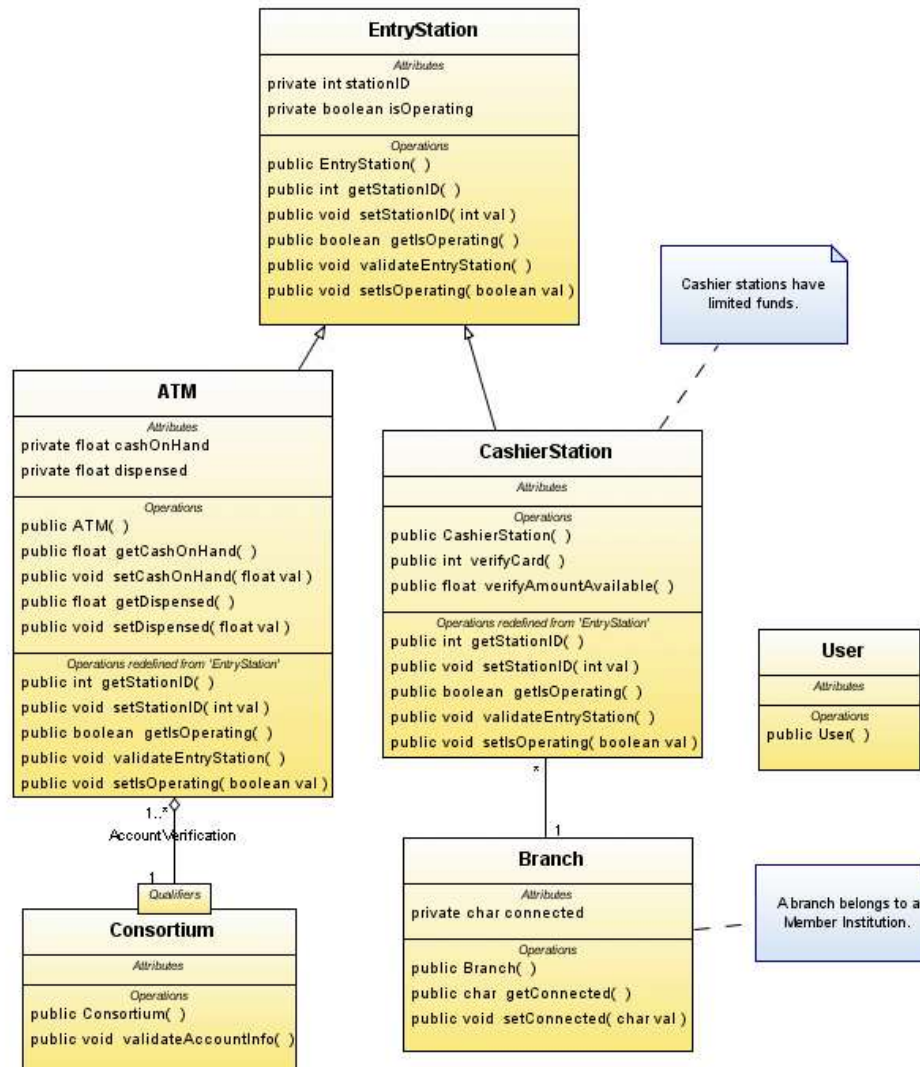
Links on associations to specify more details about the relationship

| Multiplicities                 | Meaning   |
|--------------------------------|---|
| <b>0..1</b>                    | zero or one instance. The notation <b><i>n</i> . . <i>m</i></b> indicates <b><i>n</i></b> to <b><i>m</i></b> instances. |
| <b>0..*</b> <i>or</i> <b>*</b> | no limit on the number of instances (including none).   |
| <b>1</b>                       | exactly one instance  |
| <b>1..*</b>                    | at least one instance   |

# Class Diagram Example 1



# Class Diagram Example 2

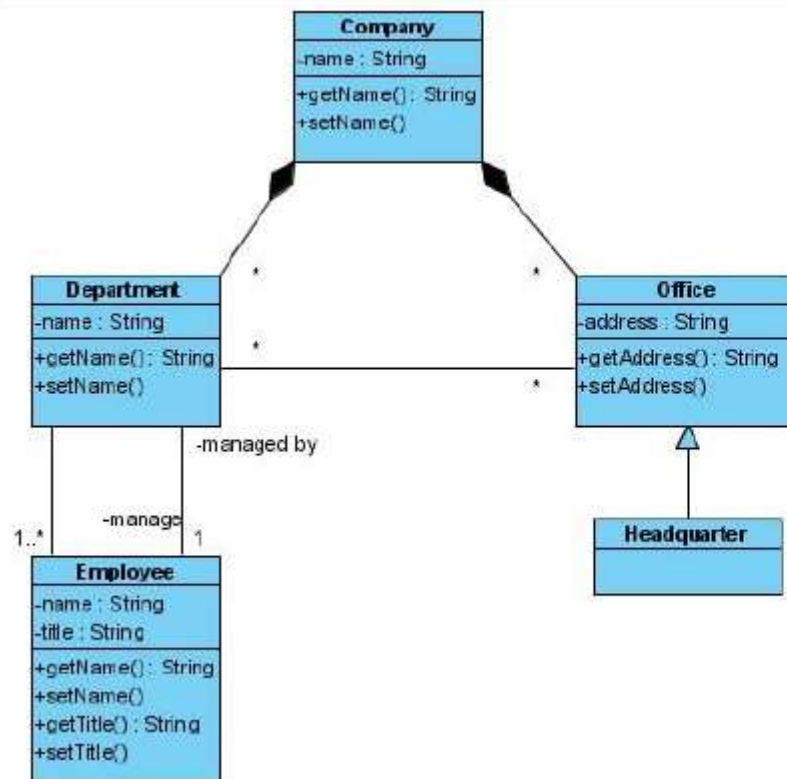


# Practice Class Diagram



- A company consists of departments. Departments are located in one or more offices. One office acts as a headquarter. Each department has a manager who is recruited from the set of employees. Your task is to model the system for the company.
- Draw a class diagram which consists of all the classes in your system their attributes and operations, relationships between the classes, multiplicity specifications, and other model elements that you find appropriate.

# Class Diagram Exercise Solution



# Composite Structure Diagram



- A Composite Structure diagram reflects the internal collaboration of Classes, Interfaces or Components (and their Properties) to describe a functionality.
- Composite Structure diagrams are similar to Class diagrams, except that they model a specific usage of the structure.

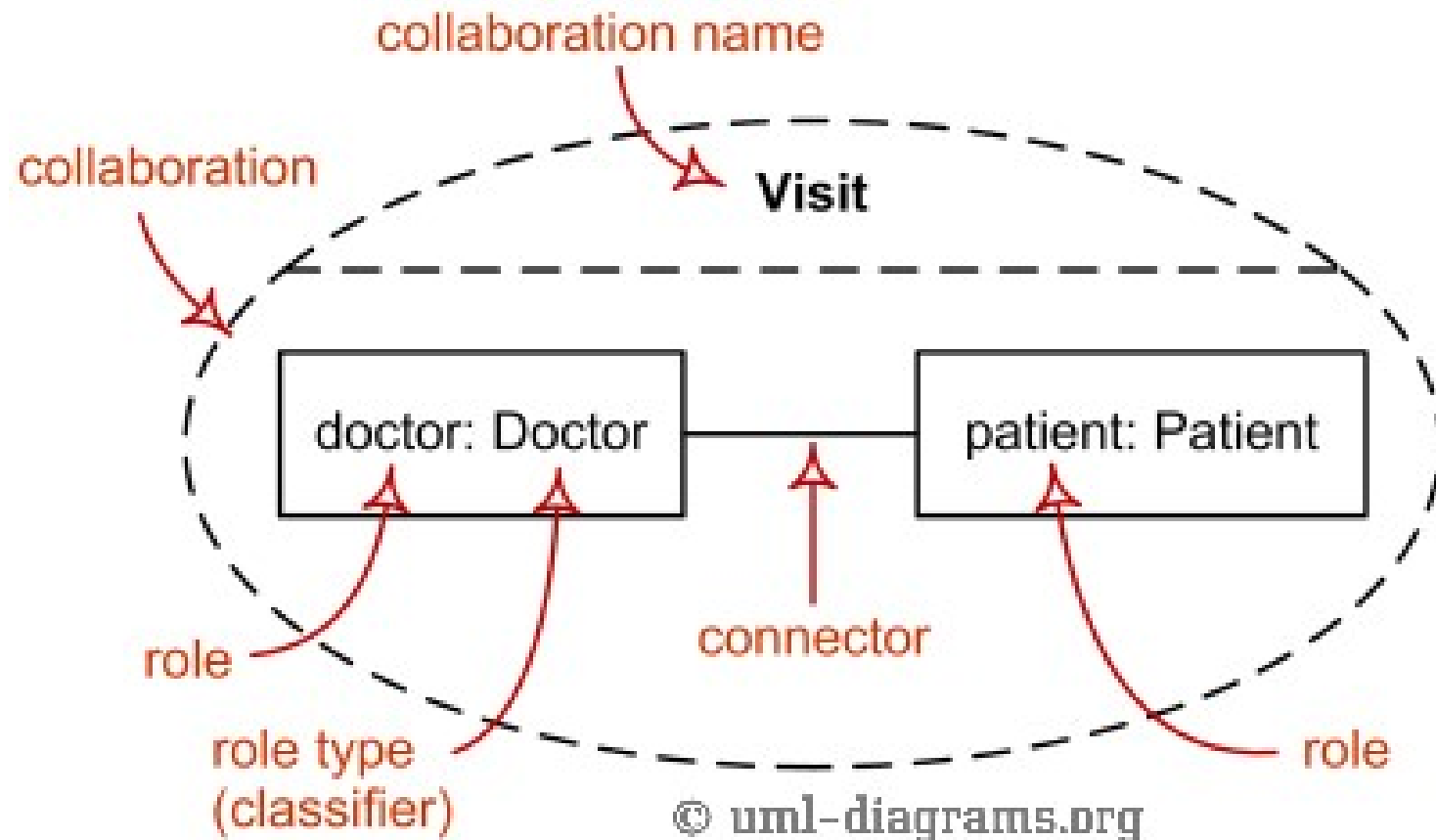
# Purpose of Composite Structure Diagram



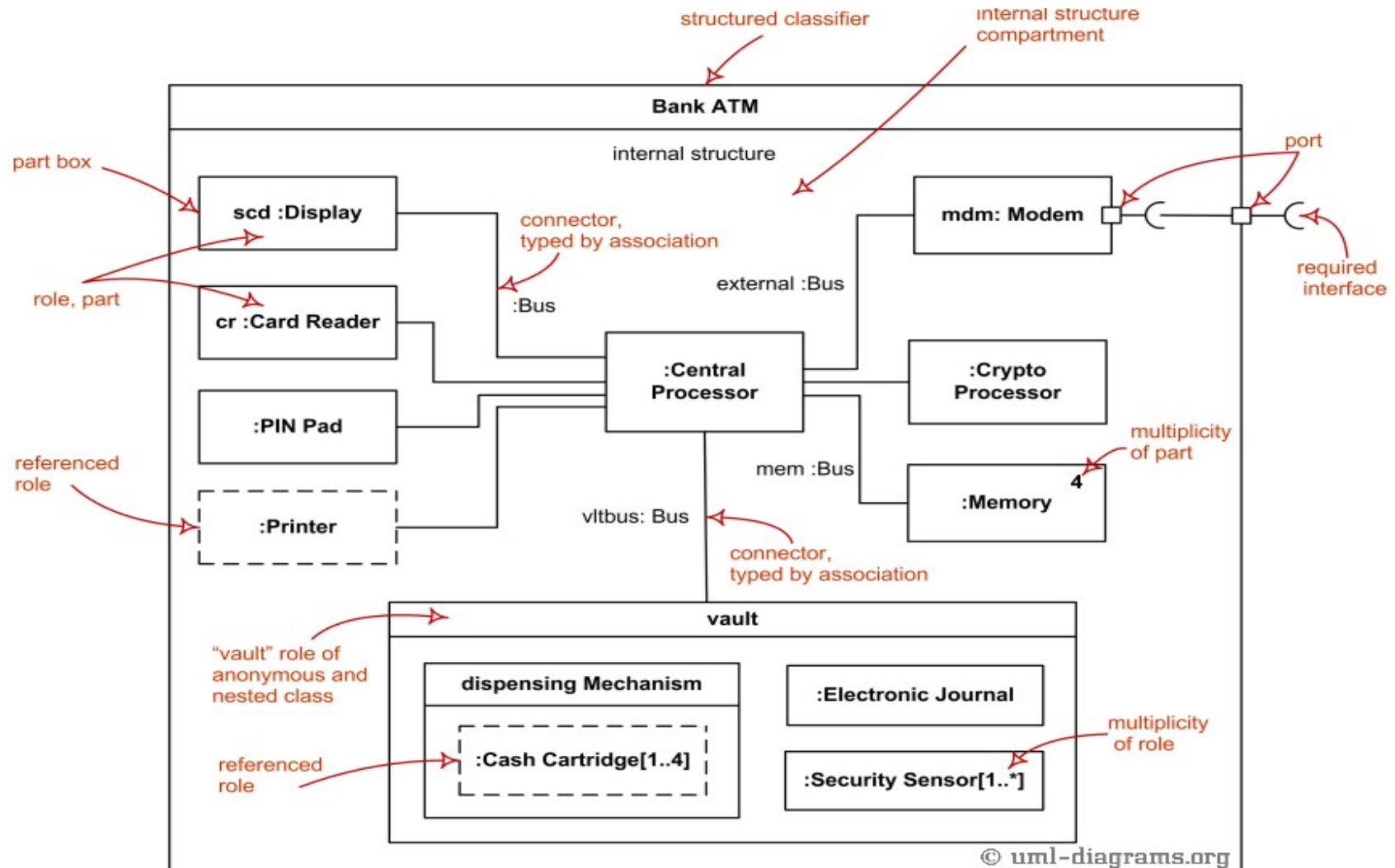
- To depict the internal structure of a classifier (such as a class, component, or use case), including the interaction points of the classifier to other parts of the system.
- To explore how a collection of cooperating instances achieves a specific task or set of tasks.
- To describe a design or architectural pattern or strategy.



# Composite Structure Diagram Example 1



# Composite Structure Diagram Example 2



# Object Diagram



- An Object diagram is closely related to a Class diagram, with the distinction that it depicts object instances of Classes and their relationships at a point in time.
- Object diagrams do not reveal architectures varying from their corresponding Class diagrams, but reflect multiplicity and the roles instantiated Classes could serve.

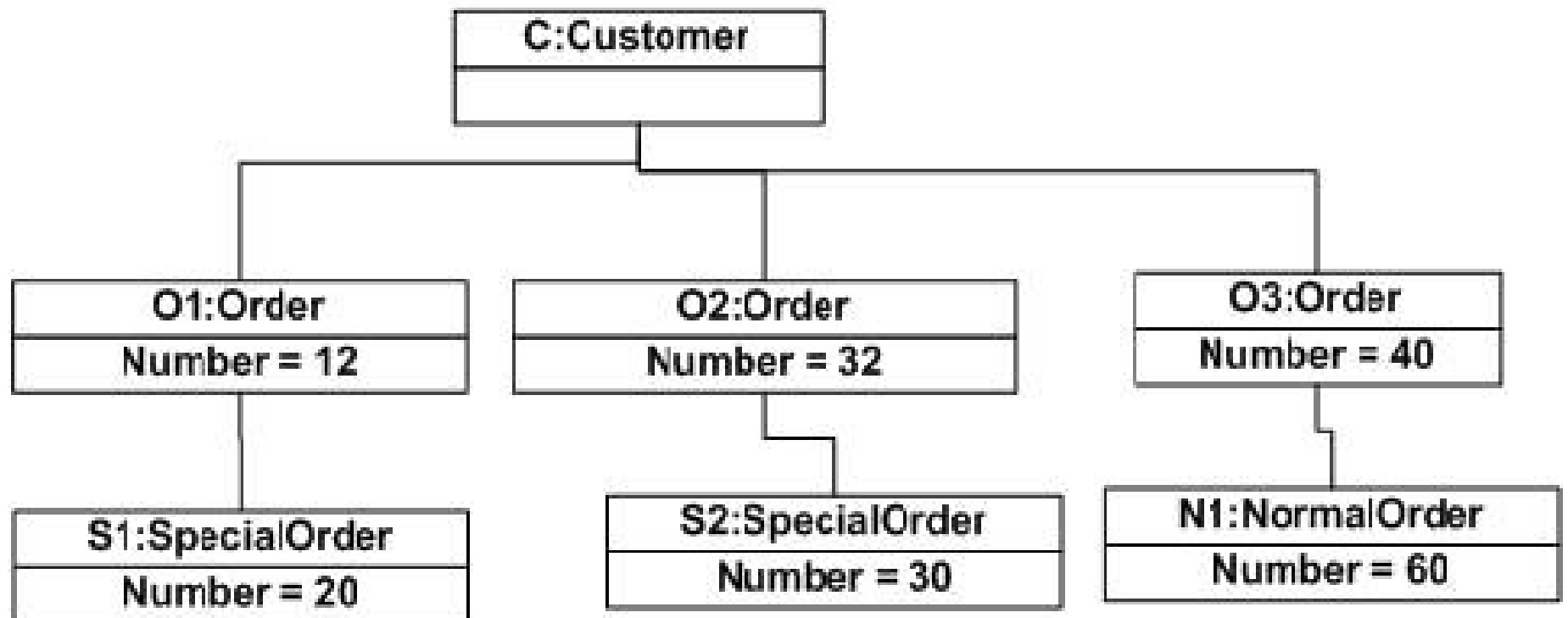
# Purpose of Object Diagram



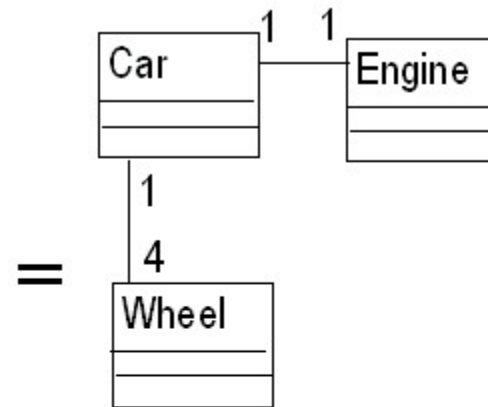
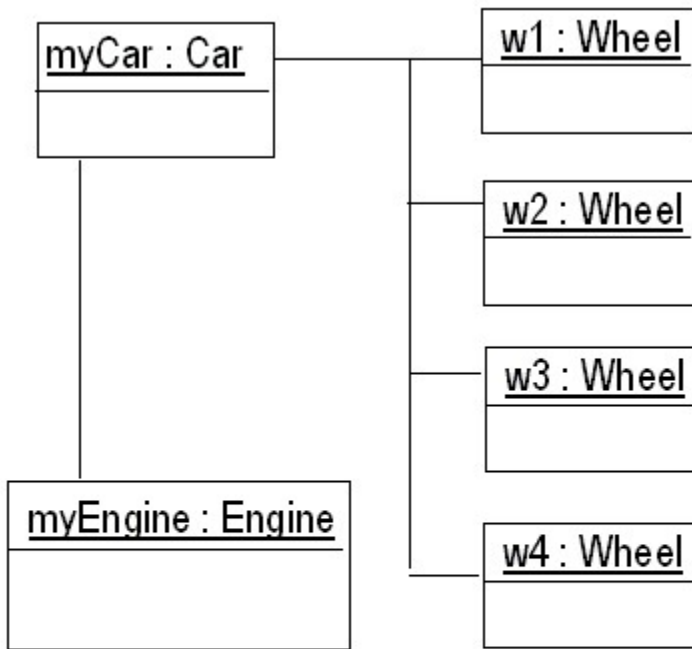
- Object diagrams are useful in understanding a complex Class diagram, by creating different cases in which the relationships and Classes are applied
- An Object diagram can also be a kind of Communication diagram (which also models the connections between objects, but additionally sequences events along each path)

# Object Diagram Example 1

Object diagram of an order management system



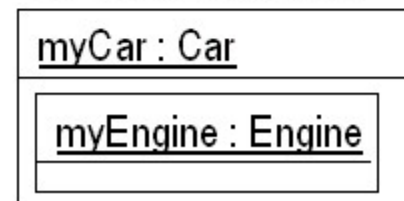
# Object Diagram Example 2



Composition with :



Or containment :



# Component Diagrams



- Shows various components in a system and their dependencies, interfaces
- Explains the structure of a system
- Usually a physical collection of classes
  - ▣ Similar to a Package Diagram in that both are used to group elements into logical structures
  - ▣ With Component Diagrams all of the model elements are private with a public interface whereas Package diagrams only display public items.

# What is a component?



Components are not a technology. Technology people seem to find this hard to understand. Components are about how customers want to relate to software . They want to be able to buy their software a piece at a time, and to be able to upgrade it just like they can upgrade their stereo . They want new pieces to work seamlessly with their old pieces, and to be able to upgrade on their own schedule, not the manufacturer's schedule . They want to be able to mix and match pieces from various manufacturers . This is a very reasonable requirement . It is just hard to satisfy.



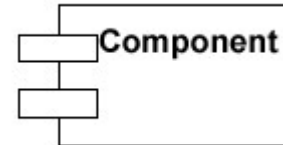
# Purpose of Component Diagram



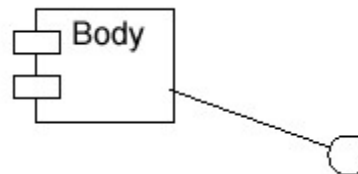
- Use component diagrams when you are dividing your System into components and want to Show their interrelationships through Interfaces or the breakdown of components into a lower-level structure.

# Component Diagram Notation

- Components are shown as rectangles with two tabs at the upper left



- Dashed arrows indicate dependencies
- Circle and solid line indicates an interface to the component



# Component Example - Interfaces

- Restaurant ordering system
- Define interfaces first – comes from Class Diagrams

○ <<user interface>>  
Order Item Form

+Begin Order()  
+Add Item()  
+Select Item()  
+Select Quantity()  
+Check Stock()  
+Enter Special Instructions()  
+Calculate Item Total()

○ <<user interface>>  
Order Confirmation Form

+Calculate Total()  
+Confirm Order()  
+Calculate Tax()  
+Calculate Restaurant Total()  
+Calculate Delivery Charge()  
+Calculate Grand Total()

○ <<user interface>>  
Error Form

+Display Error Message()

○ IOrderSystem

+Create Order()

○ IOrder

+Add Item()  
+Place Order()

○ IRestaurantSystem.

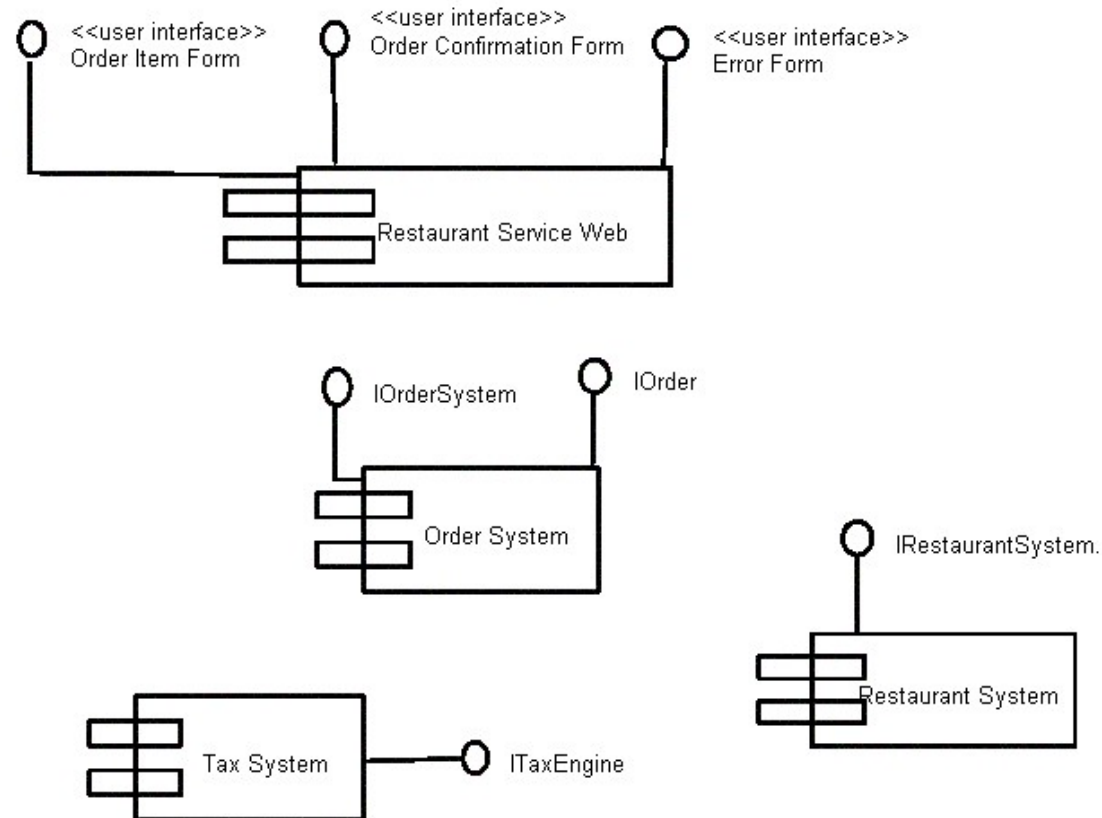
+Place Order()  
+Check Stock()

○ ITaxEngine

+Calculate()

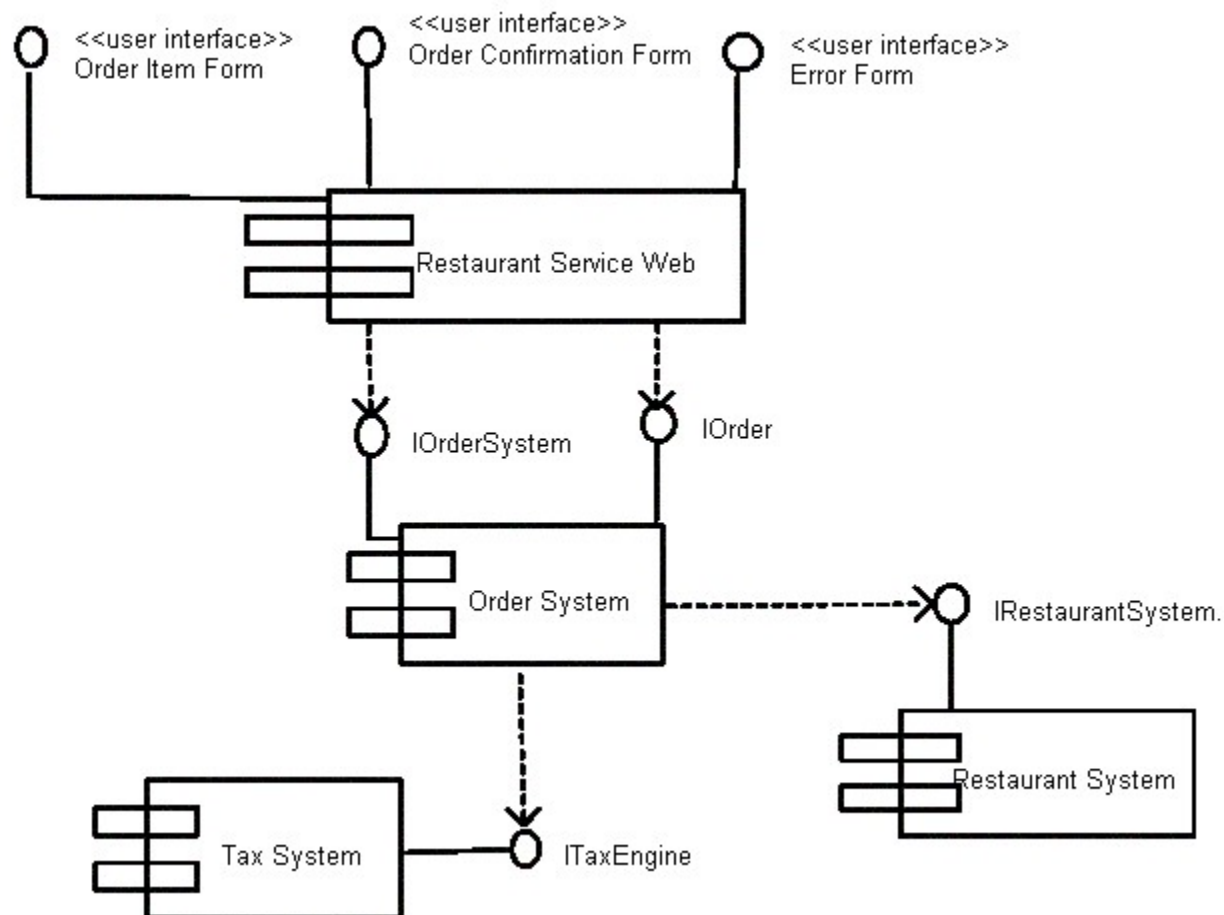
# Component Example - Components

## □ Graphical depiction of components

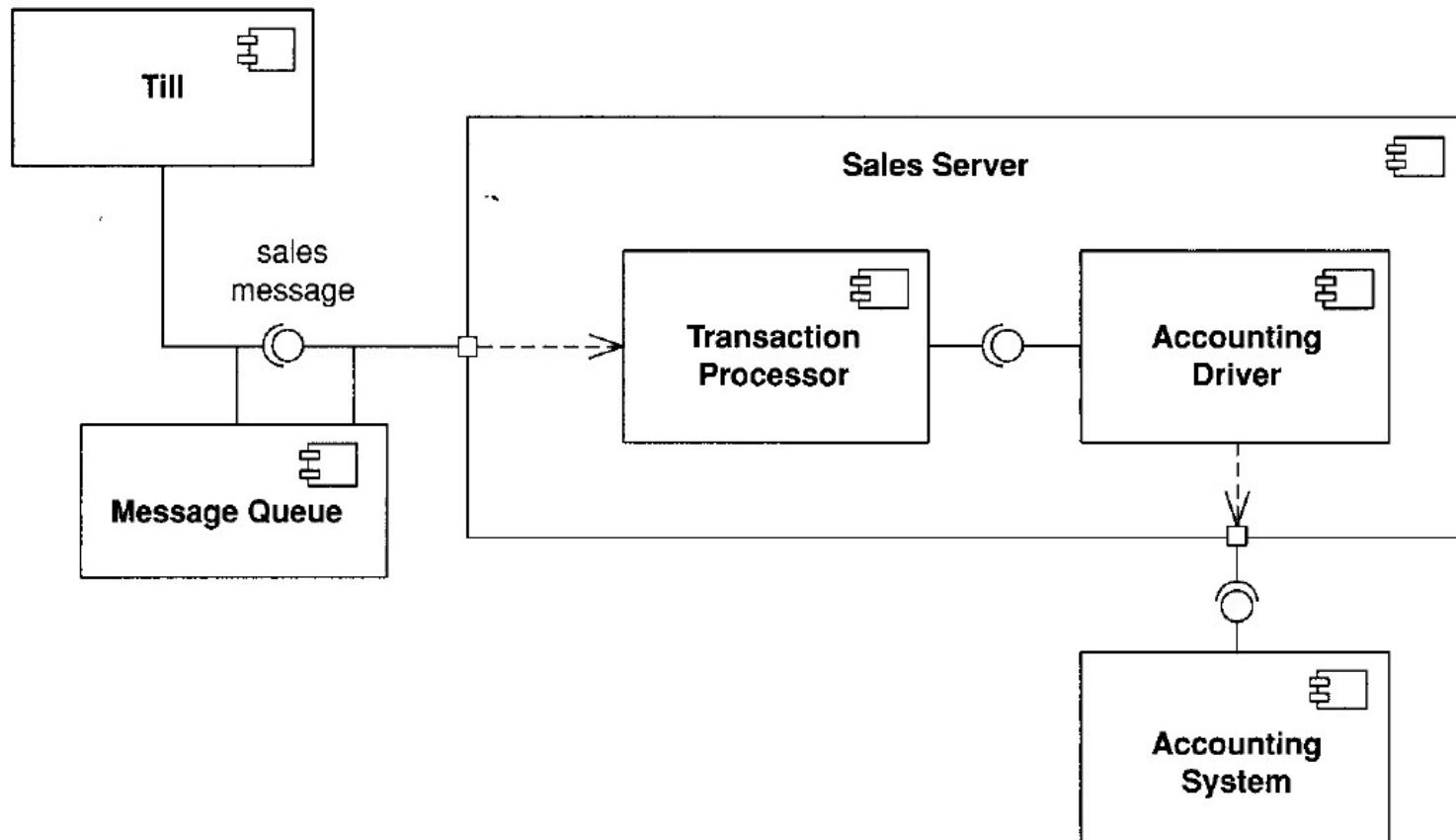


# Component Example - Linking

## □ Linking components with dependencies



# Component Diagram Example



# Practice Component Diagram

- Draw a component diagram for Ascend Banking
- Draw a component diagram for Ascend MF

# Package Diagrams

- To organize complex class diagrams, you can group classes into packages. A package is a collection of logically related UML elements
- Notation
  - ▣ Packages appear as rectangles with small tabs at the top.
  - ▣ The package name is on the tab or inside the rectangle.
  - ▣ The dotted arrows are dependencies. One package depends on another if changes in the other could possibly force changes in the first.
  - ▣ Packages are the basic grouping construct with which you may organize UML models to increase their readability

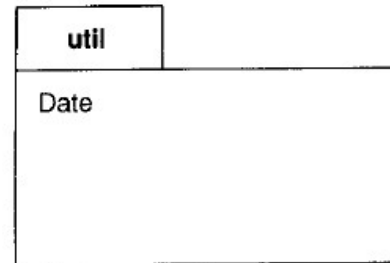
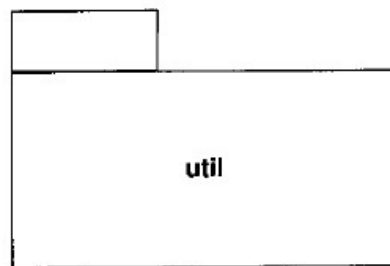


# Purpose of Package Diagram

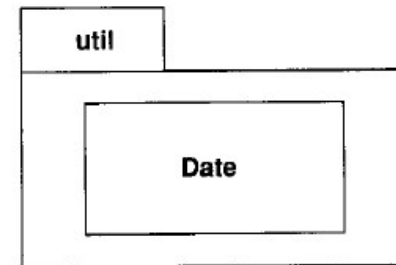


- To depict the organization of model elements into packages
- Show dependencies between packages
- Visualize namespaces
- On larger-scale systems to get a picture of the dependencies between major elements of a System
- Package diagrams represent a compile-time grouping mechanism
- Most commonly used for showing the grouping of classes

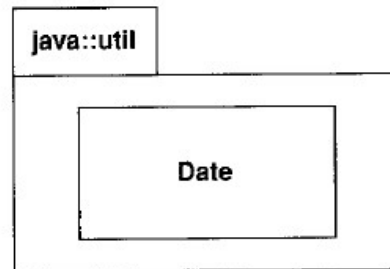
# Ways of Showing Packages



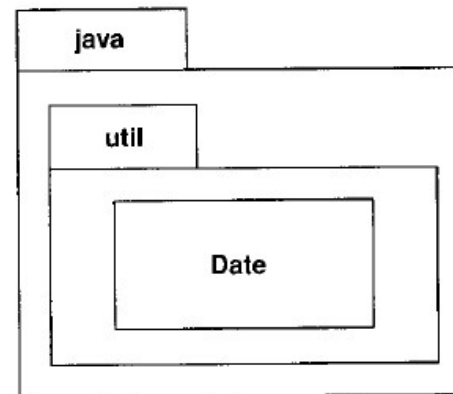
Contents listed in box



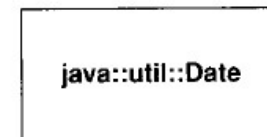
Contents diagrammed in box



Fully qualified package name

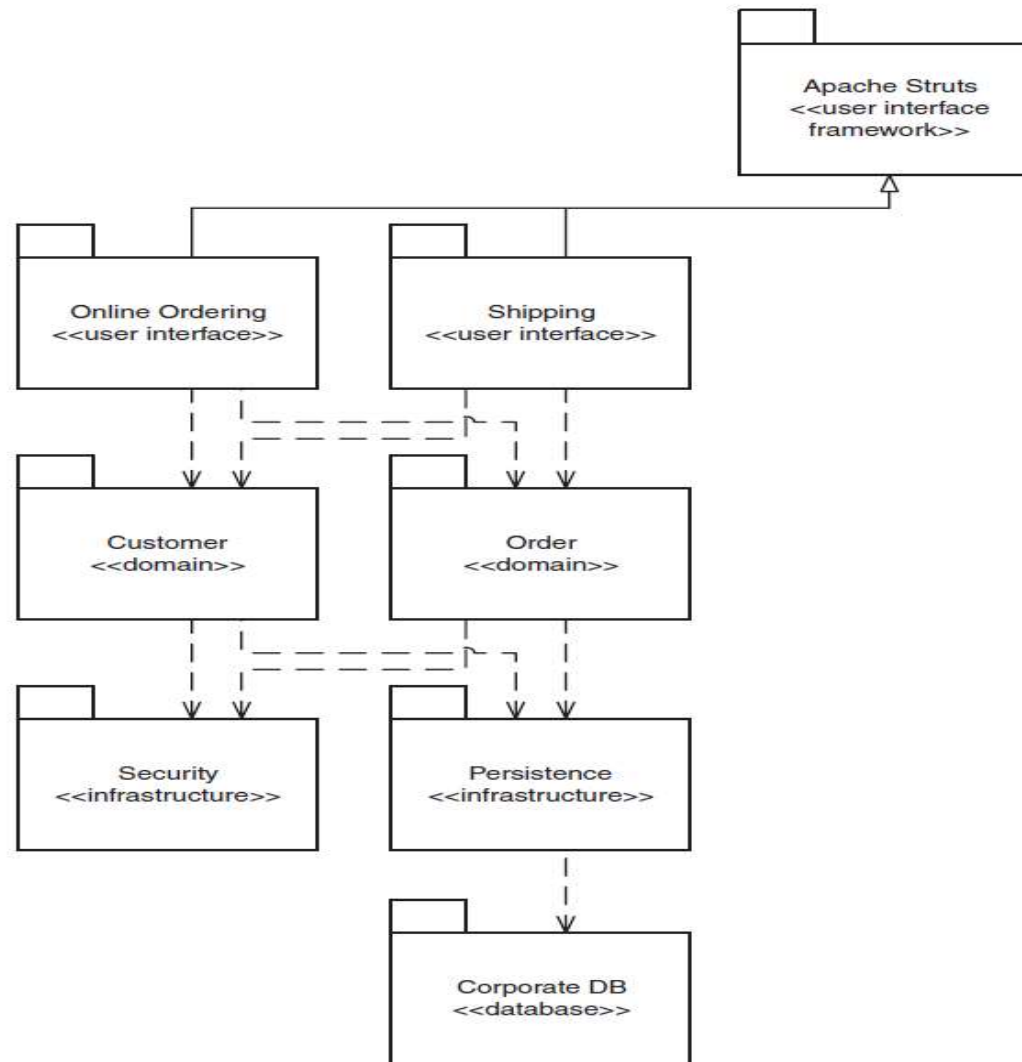


Nested packages

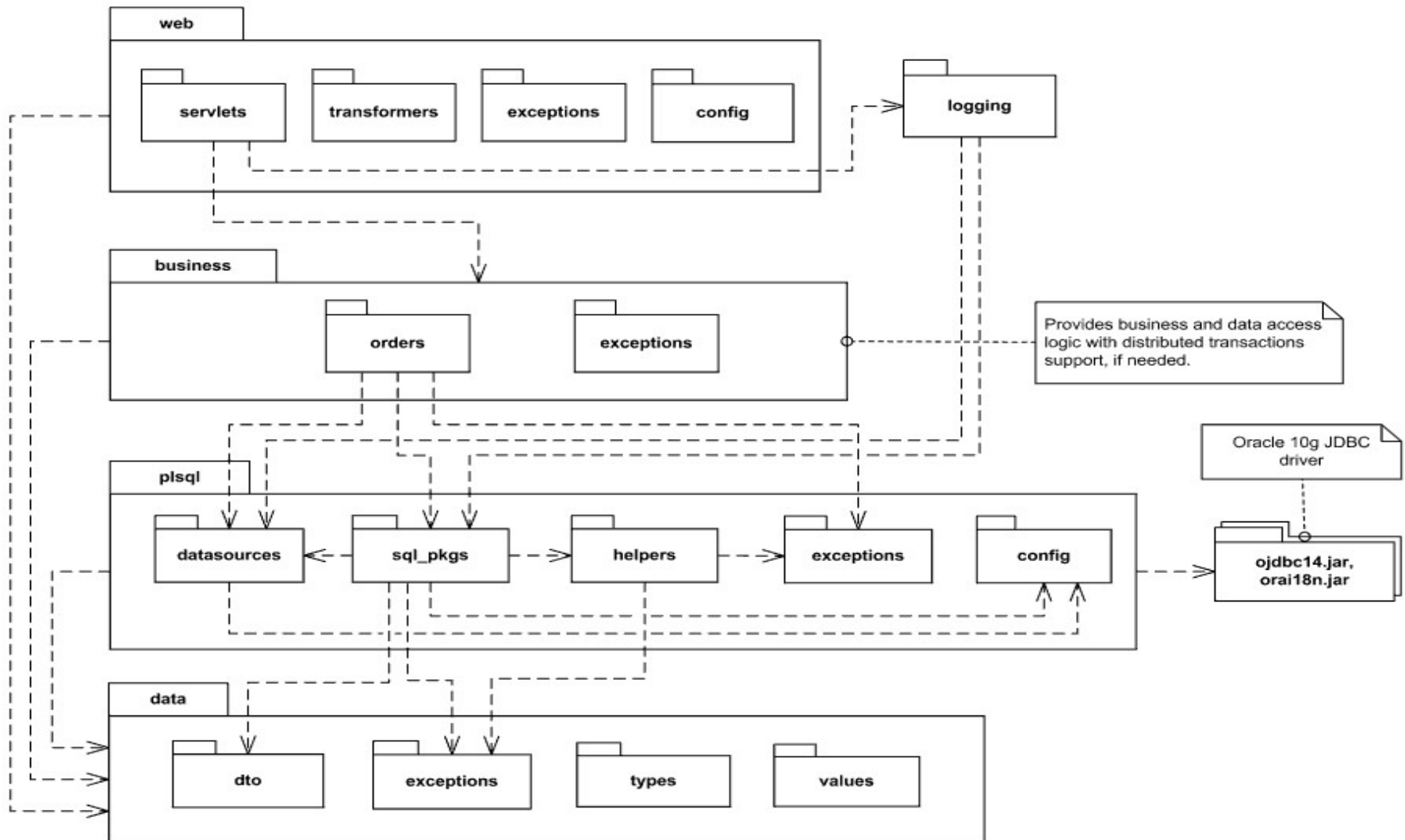


Fully qualified class name

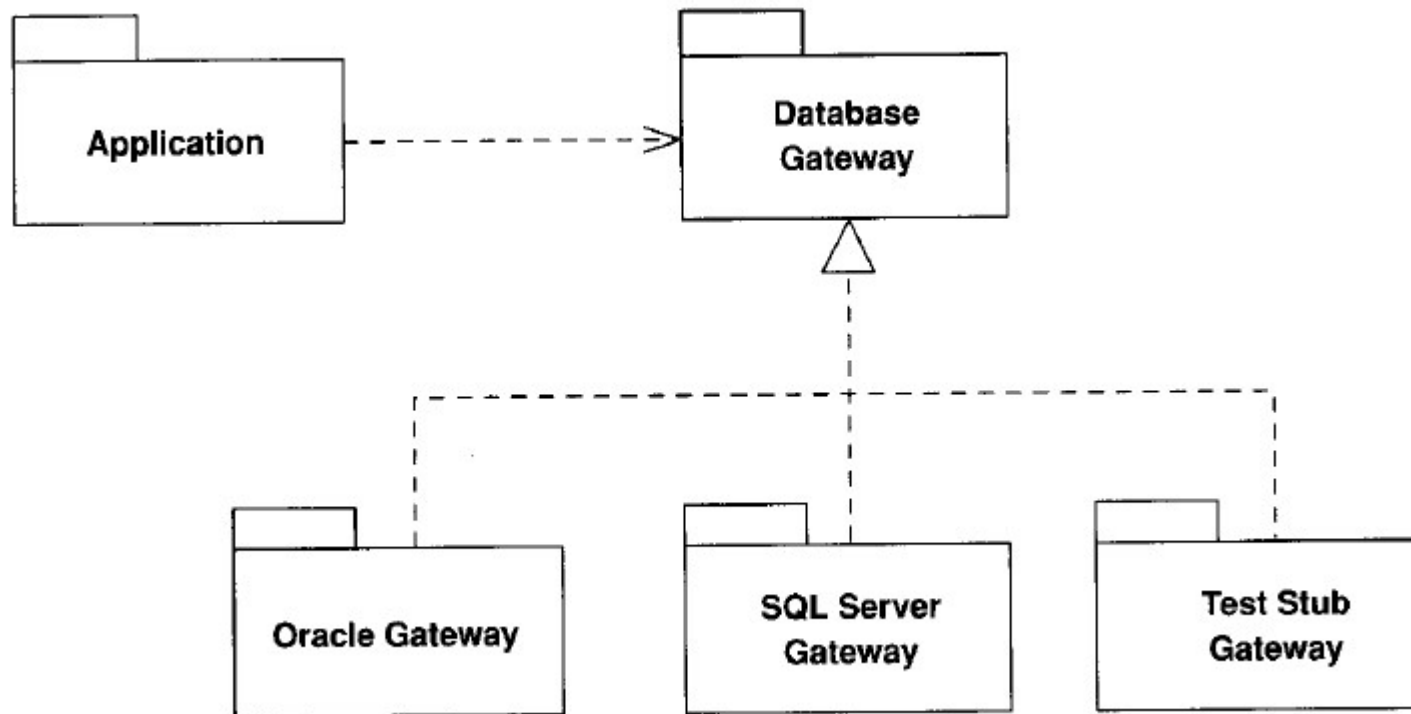
# Package Diagram Example 1



# Package Diagram Example 2



# Package Diagram Example 3



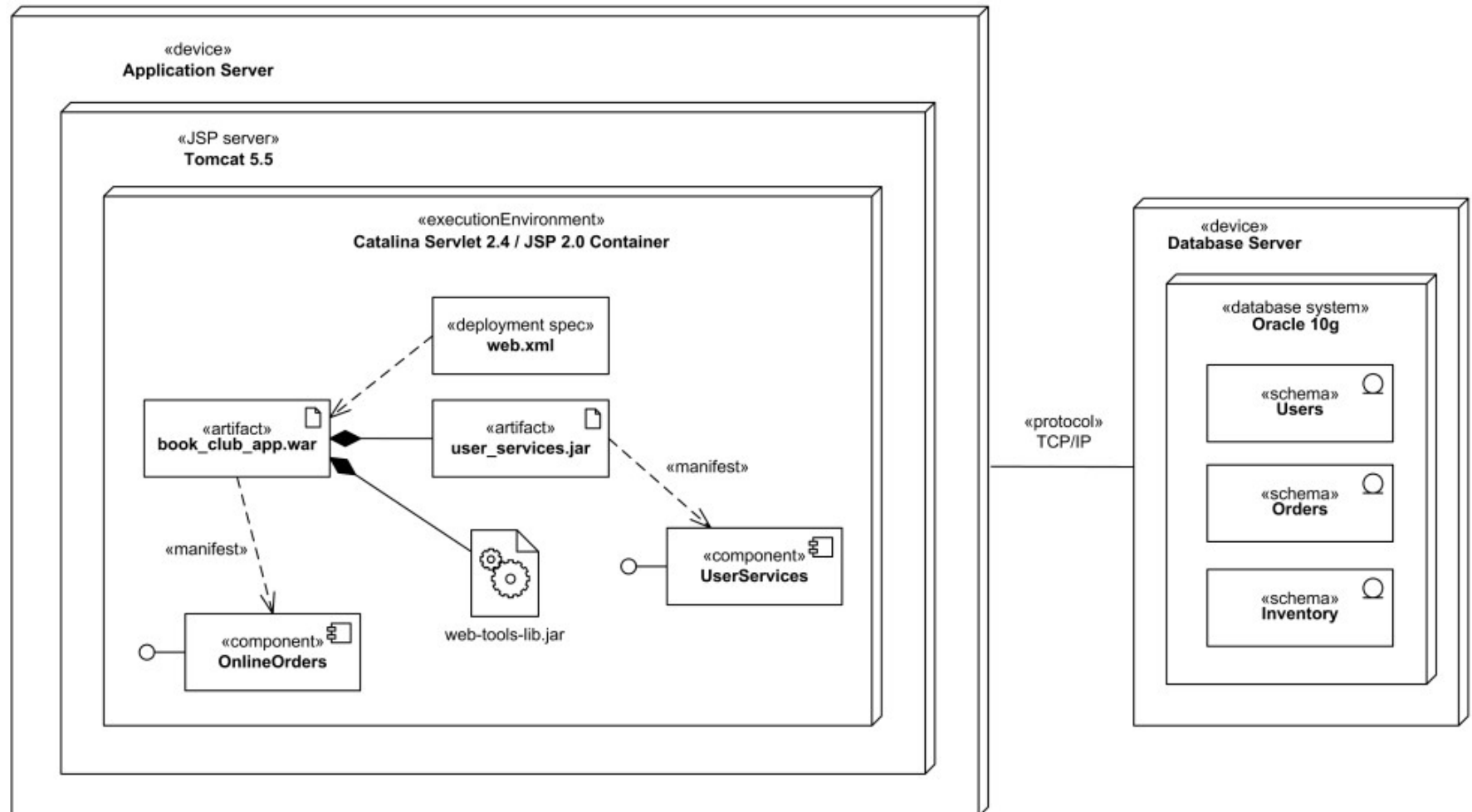
# Practice Package Diagram

- Draw a package diagram for Ascend Banking
- Draw a package diagram for Ascend MF

# Deployment Diagrams

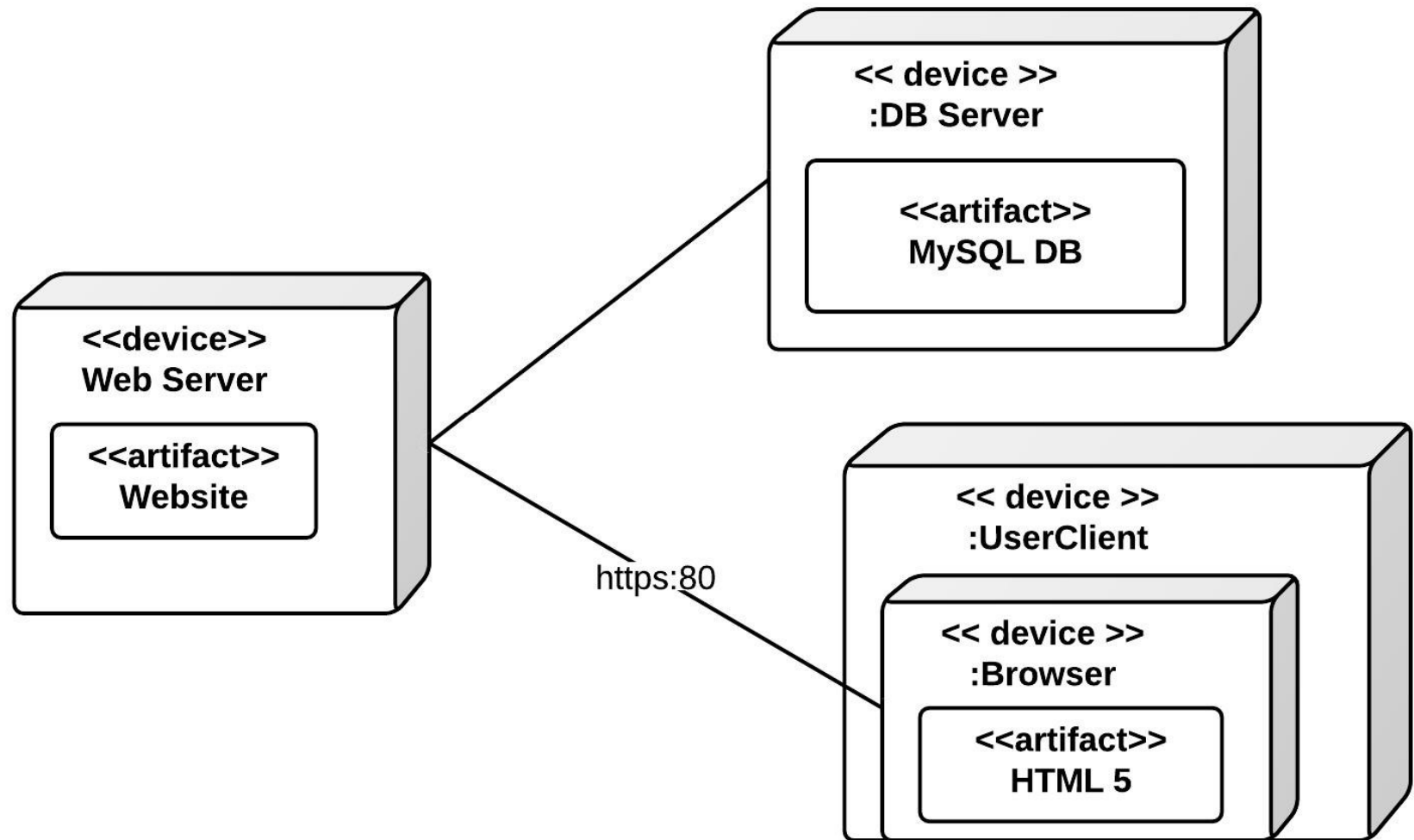
- Shows the physical architecture of the hardware and software of the deployed system
- Nodes
  - ▣ Typically contain components or packages
  - ▣ Usually some kind of computational unit; e.g. machine or device (physical or logical)
- Physical relationships among software and hardware in a delivered systems
  - ▣ Explains how a system interacts with the external environment

# Deployment Diagram Example 1

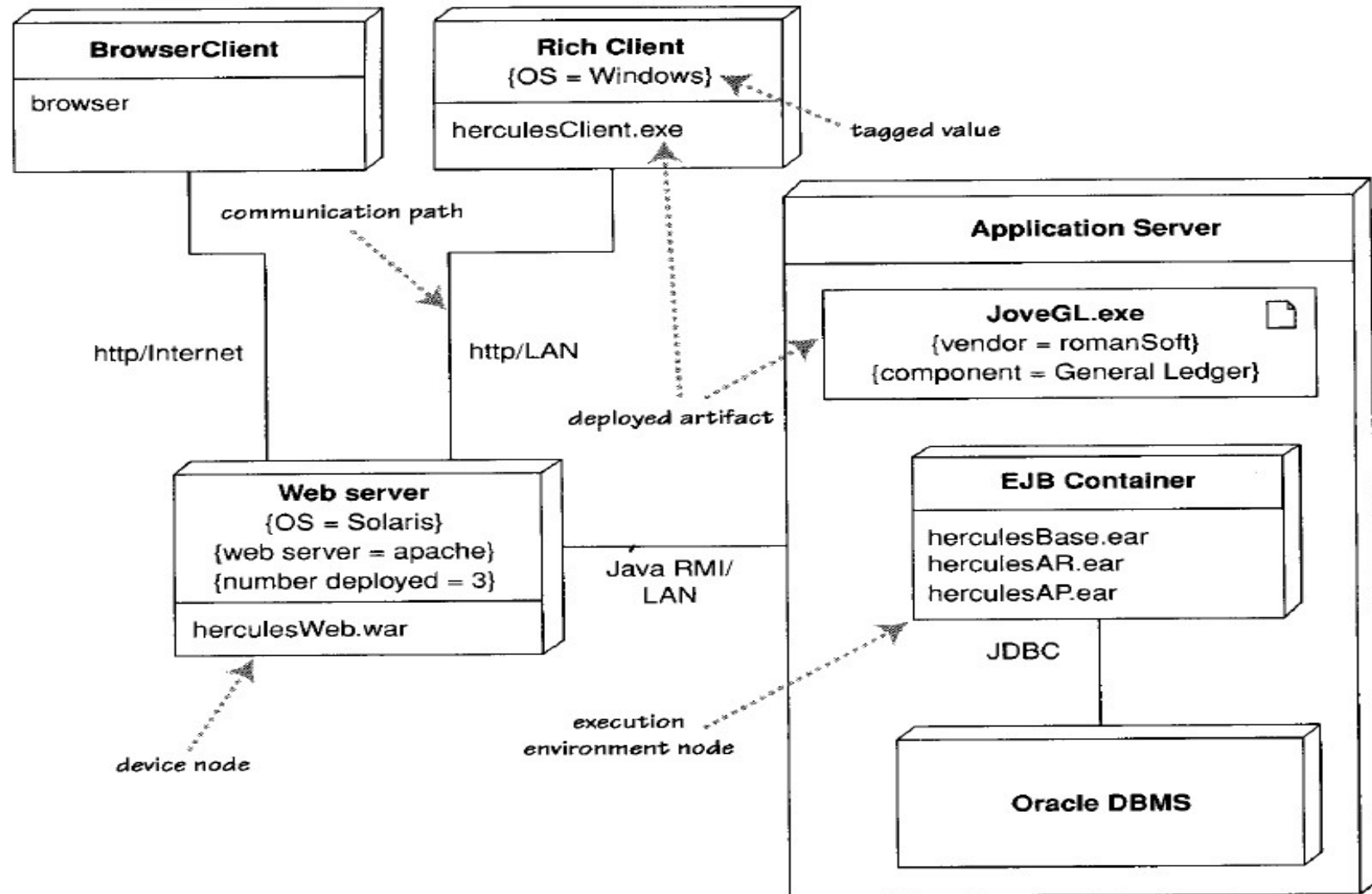




# Deployment Diagram Example 2



# Deployment Diagram Example 3



END OF STRUCTURAL DIAGRAM

# Recommended Books



- The Elements of UML 2.0 Style
- UML Distilled

# Reference



- <http://www.tutorialspoint.com/uml/index.htm>
- [http://en.wikipedia.org/wiki/Unified\\_Modeling\\_Language](http://en.wikipedia.org/wiki/Unified_Modeling_Language)
- <http://www.uml.org/>
- <http://www.uml-diagrams.org/>
- [http://www.sparxsystems.com.au/resources/uml2\\_tutorial/](http://www.sparxsystems.com.au/resources/uml2_tutorial/)

**THANK YOU**

