

Introduction to UML

Presenter: Longbin Lai

longbin.lai@gmail.com

As a User



User: Hey, Database system, please give me the documents bla bla bla

DBMS: Your majesty, I can find everything you need, if I not, I can Google it for you.



As an Architecture ...

Before proposing the project

What's the user cases?

How many users are there?

While designing the system

How many components? What does each do?

What objects and relationships to involve?

What attributes are there? (e.g. ER model)

While implementing the system

What activities one object gonna involve in an event?

When will one object hold certain resource, for how long?

How states change in response to the events?



As an Architecture ...

????????????????????
????????????????????



We need a **modelling language** that can model:

User Cases

Objects and their relationships

The interactions of objects in the process

The sequence of objects in the process

The states of objects in the process

The activities that objects gonna involve

.....

What is UML?

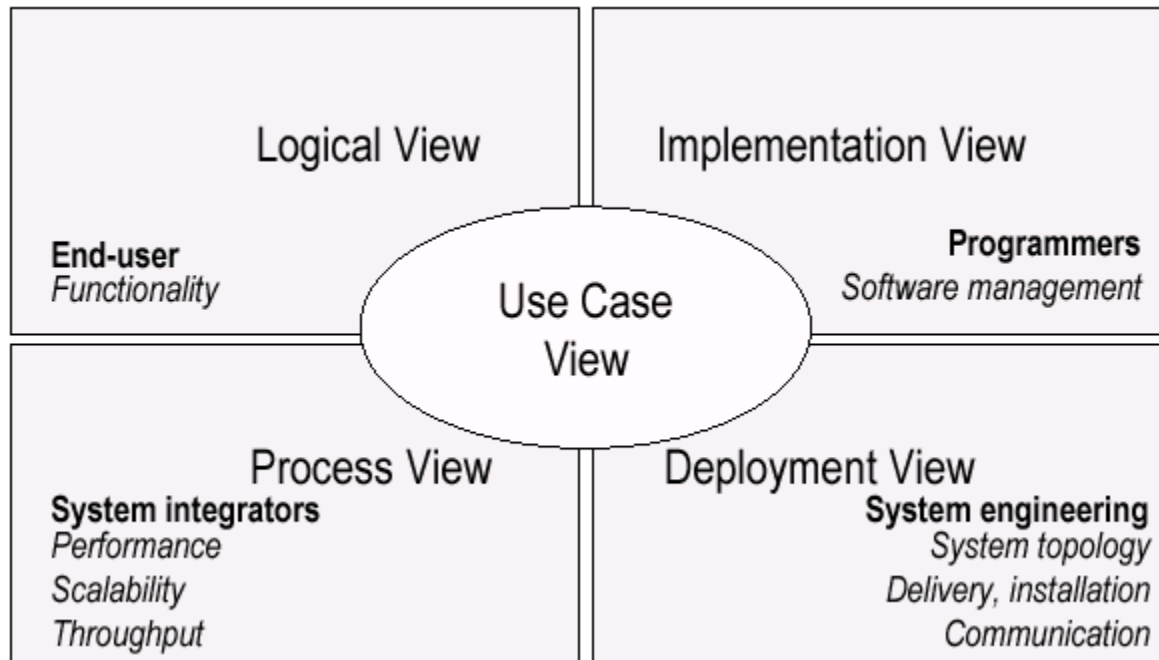
- Unified Modelling Language
 - OMG Standard, Object Management Group
 - Based on the works of Booch, Rumbaugh, Jacobson
- UML is a modelling language to express and design documents, software
 - Particularly useful for object-oriented design
 - Not a process, but some have been proposed using UML
 - Independent of programming language

Why use UML

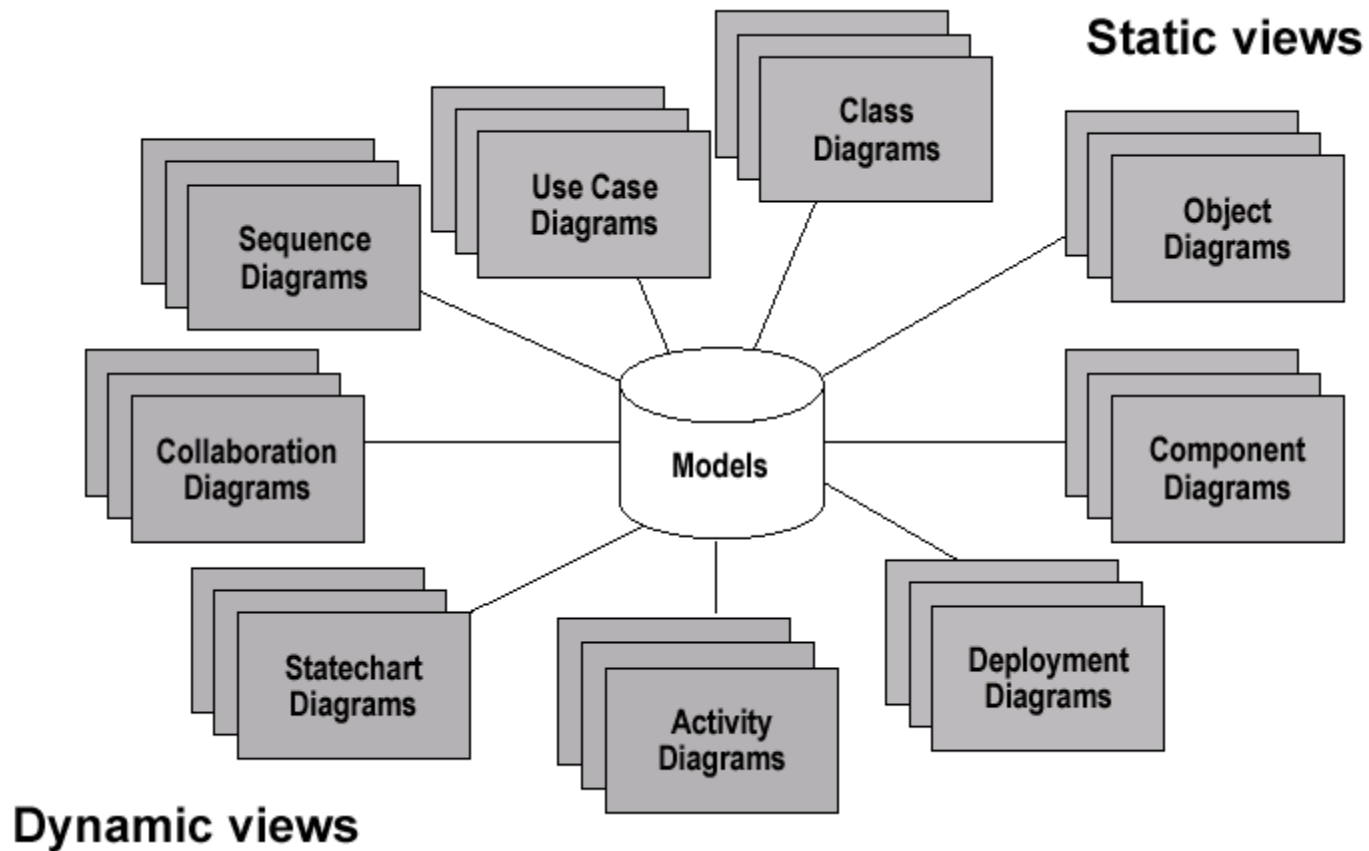
- Open Standard, Graphical notation for
 - Specifying, visualising, constructing, and documenting software systems
- Increase understanding/communication of product to customers and developers
- Support for diverse application areas
- Support for UML in many software packages today
- Based upon experience and needs of the user community

UML Views, Diagrams

- UML is a multi-diagrammatic language
 - Each diagram is a view into a (part of) the system
 - Diagram presented from the aspect of a particular subject
 - Provides a partial representation of the system
 - Is semantically consistent with other views
 - Example views



UML Views, Diagrams



How Many Views?

- Views should to fit the context
 - Not all systems require all views
 - Single processor: drop deployment view
 - Single process: drop process view
 - Very small program: drop implementation view
- A system might need additional views
 - Data view, security view, ...

UML: First Pass

- You can model 80% of most problems by using about 20 % UML
- We only cover the 20% here

Basic Modelling Steps

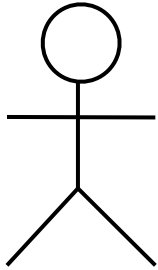
- Use Cases
 - Capture requirements
- Domain Model
 - Capture process, key classes (entities), and their relationships
- Design Model
 - Capture details and behaviours of use cases and domain objects
 - Add classes that do the work and define the architecture

Basic Models

- **Use Case Diagrams**
- **Class Diagrams**
- **Package Diagrams**
- **Interaction Diagrams**
 - Sequence
 - Collaboration
- **Activity Diagrams**
- **State Transition Diagrams**
- **Deployment Diagrams**

Use Case Diagrams

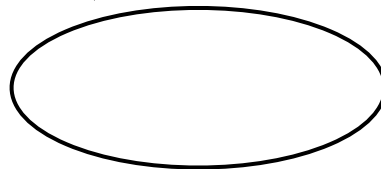
Use Case Diagrams



Passenger

- **Actors** represent roles, that is, a type of user of the system

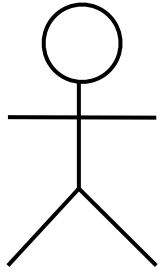
- **Use cases** represent a sequence of interaction for a type of functionality; summary of scenarios



PurchaseTicket

- The **use case model (diagram)** is the set of all use cases. It is a complete description of the functionality of the system and its environment

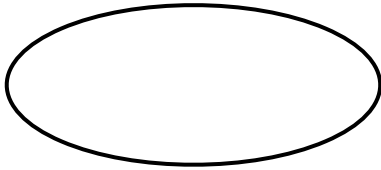
Actors



Passenger

- 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 Diagram: 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.

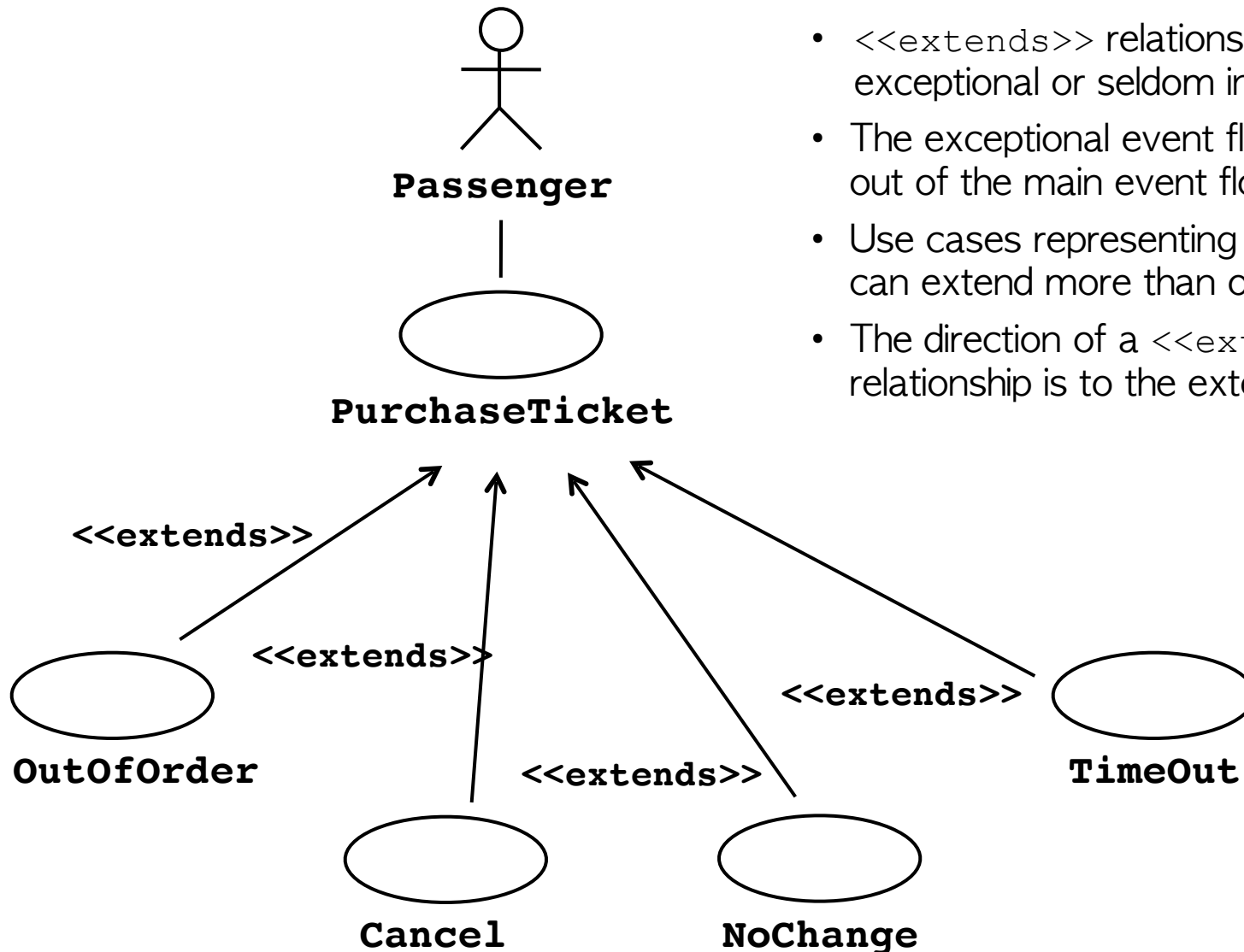
Event flow:

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?

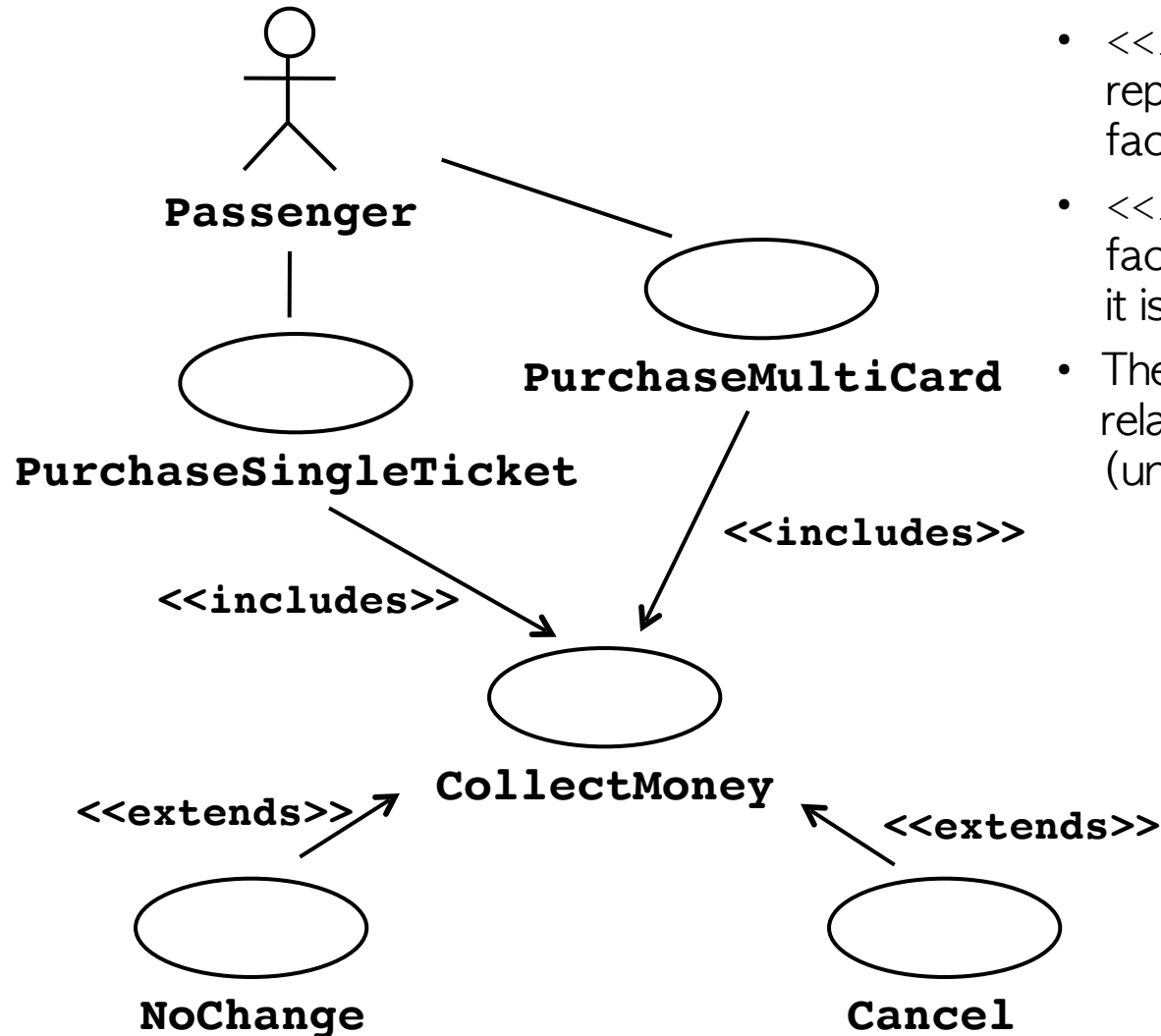
Exceptional cases!

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 behaviour that is factored out of the use case.
- <<includes>> behaviour 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).

Use Cases are useful to...

- Determining requirements
 - New use cases often generate new requirements as the system is analysed 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.

Use Case Diagrams: Summary

- Use case diagrams represent external behaviour
- Use case diagrams are useful as an index into the use cases
- All use cases need to be described for the model to be useful
- Use case diagrams can guide the test inputs

Class Diagrams

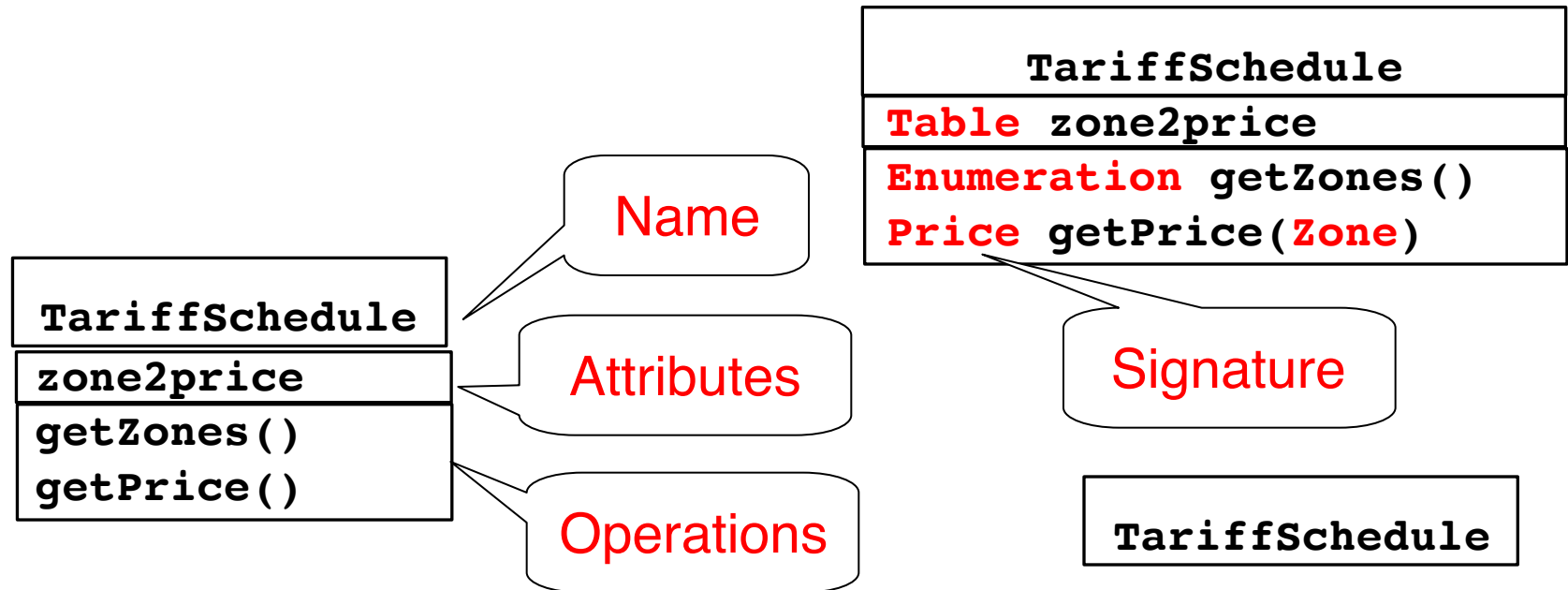
Class Diagrams

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

Class Diagram Perspectives

- We draw Class Diagrams on three levels
 - Conceptualisation
 - 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 behaviour (*operations*).
- Each attribute has a *type*.
- Each operation has a *signature* (*return type*).
- The class name is the only mandatory information.

Instances

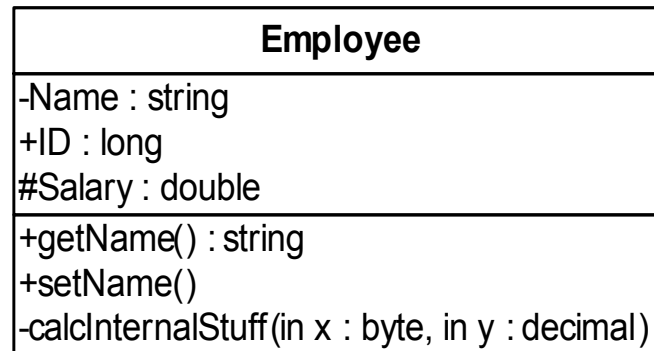
```
tarif_1974:TariffSchedule
```

```
zone2price = {  
    {'1', .20},  
    {'2', .40},  
    {'3', .60}}
```

- An *instance* represents an object of the class.
- The name of an instance is underlined and can contain the class of the instance.
- The attributes are represented with their *values*.

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

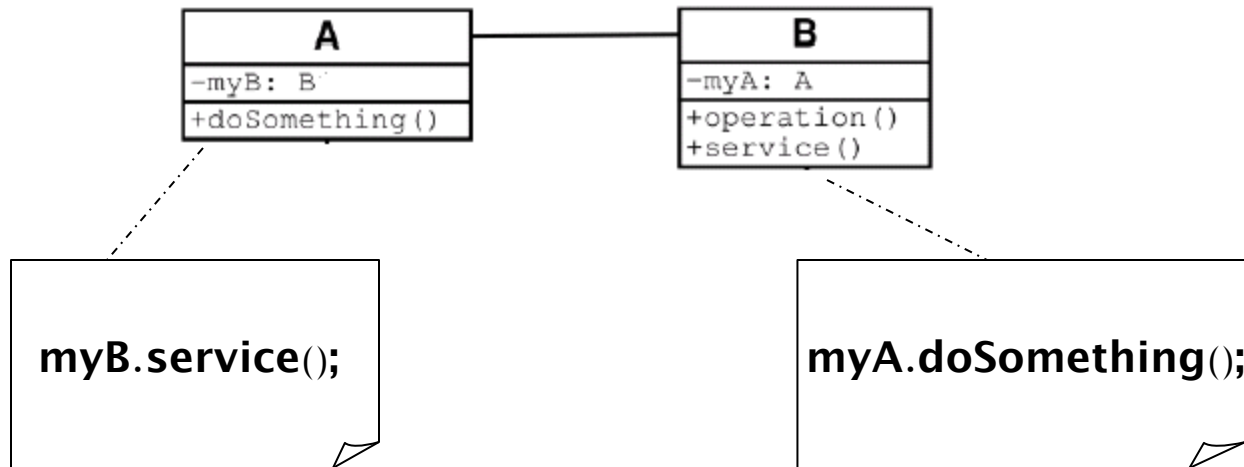


UML Class Notation

- 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. a member of a team
 - 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. monkey is a kind of mammal
 - Indicated by triangle pointing to superclass

Binary Association

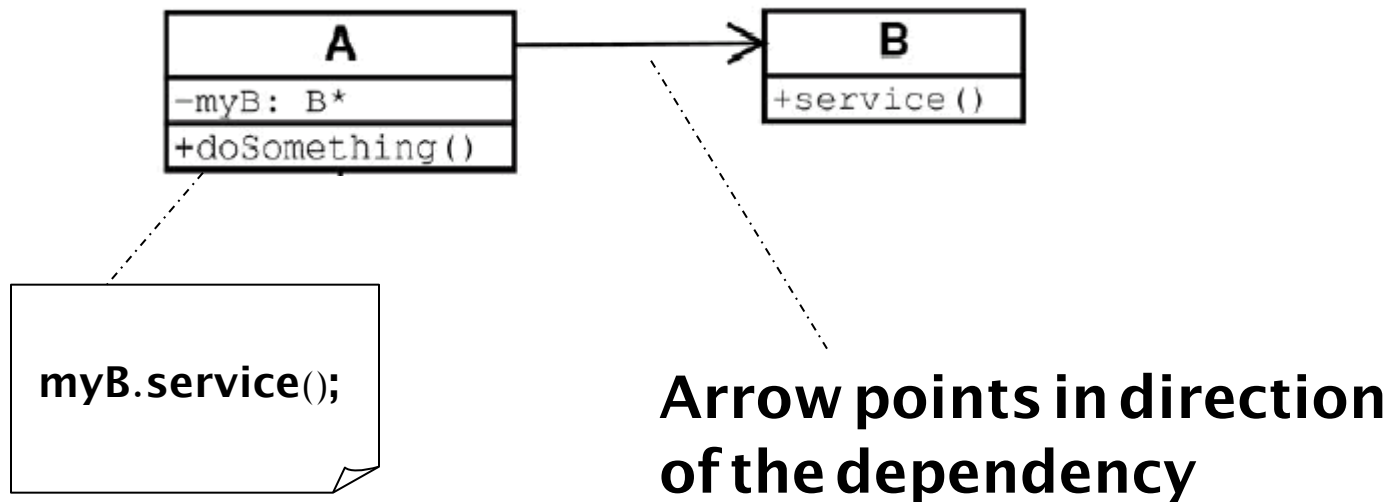
Binary Association: Both entities “Know About” each other



Optionally, may create an Associate Class

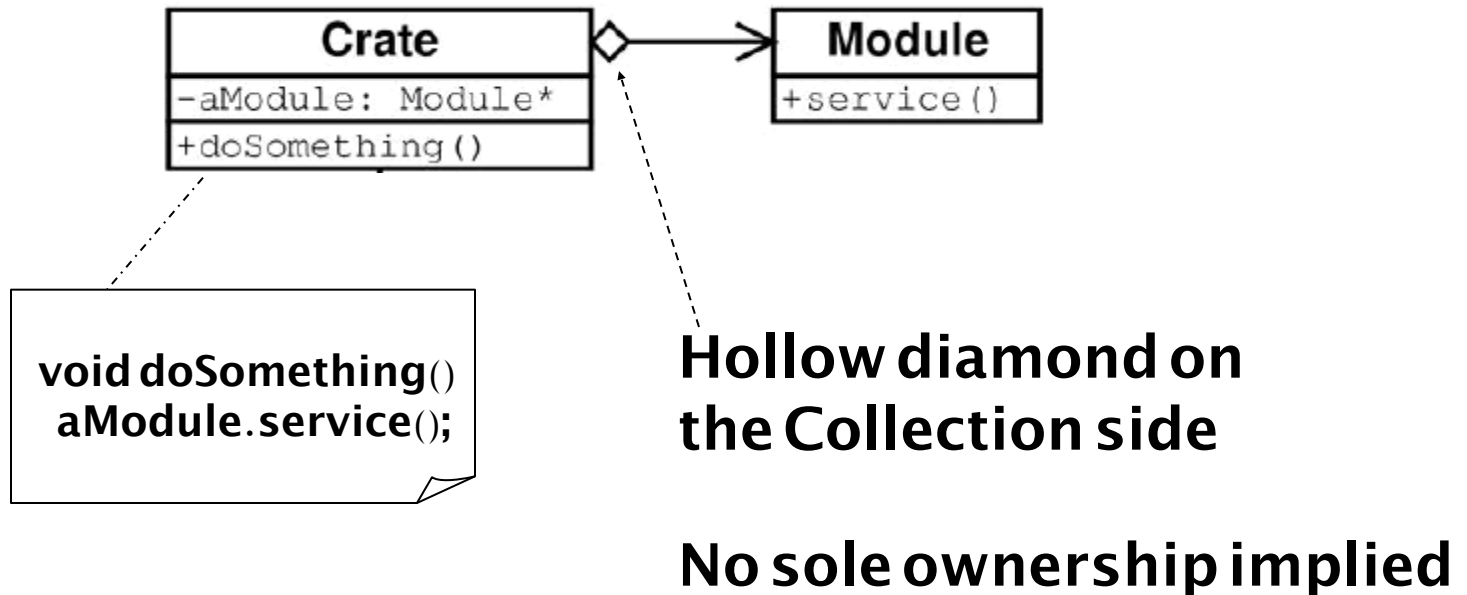
Unary Association

A knows about B, but B knows nothing about A



Aggregation

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

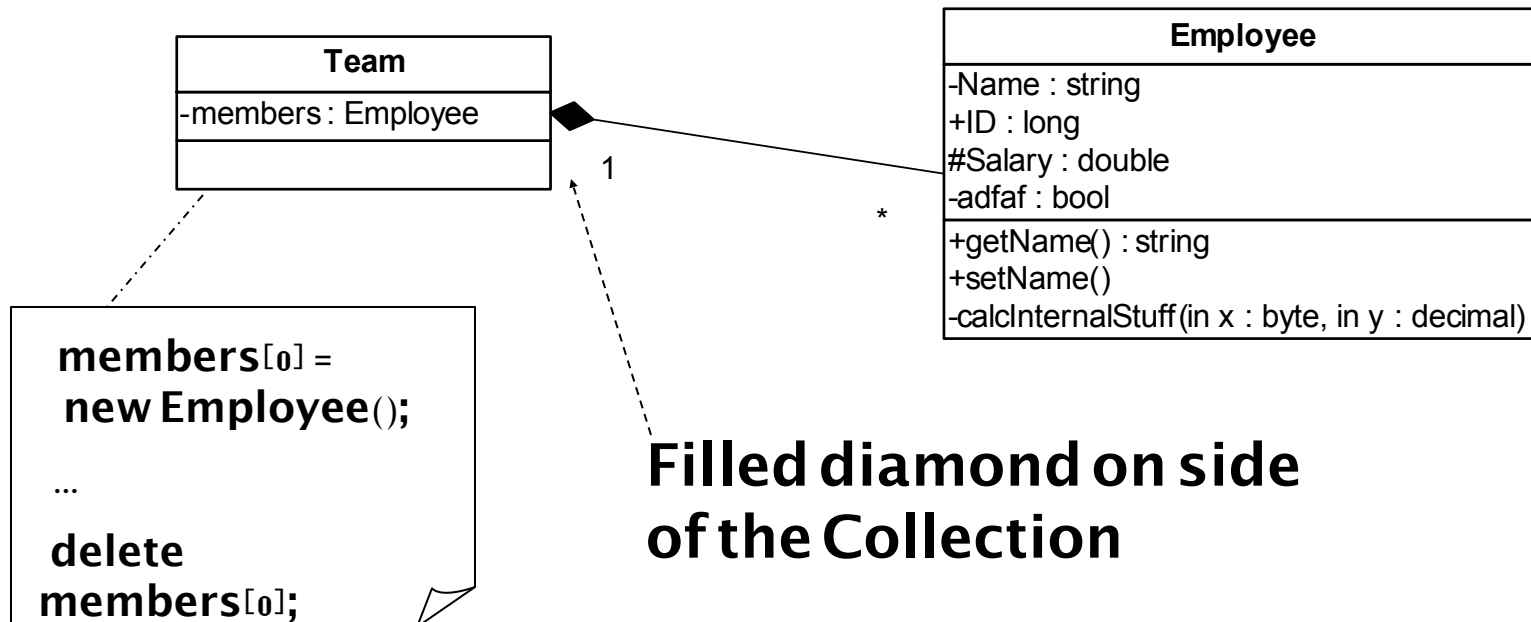


Composition

Composition is Aggregation with:

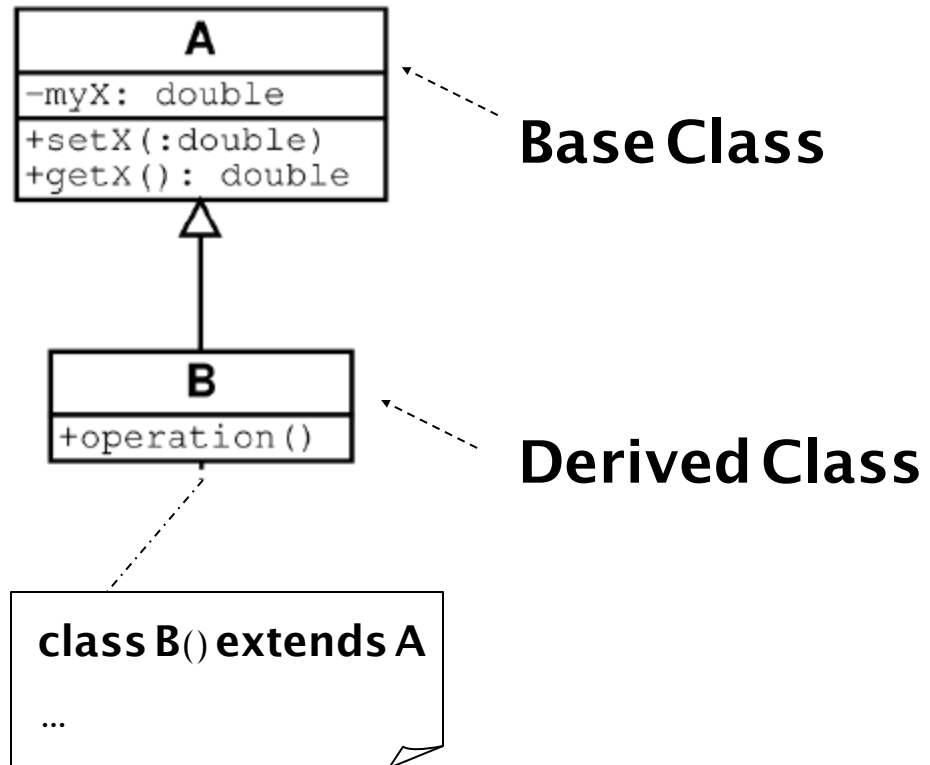
Lifetime Control (owner controls construction, destruction)

Part object may belong to only one whole object



Inheritance

Standard concept of inheritance

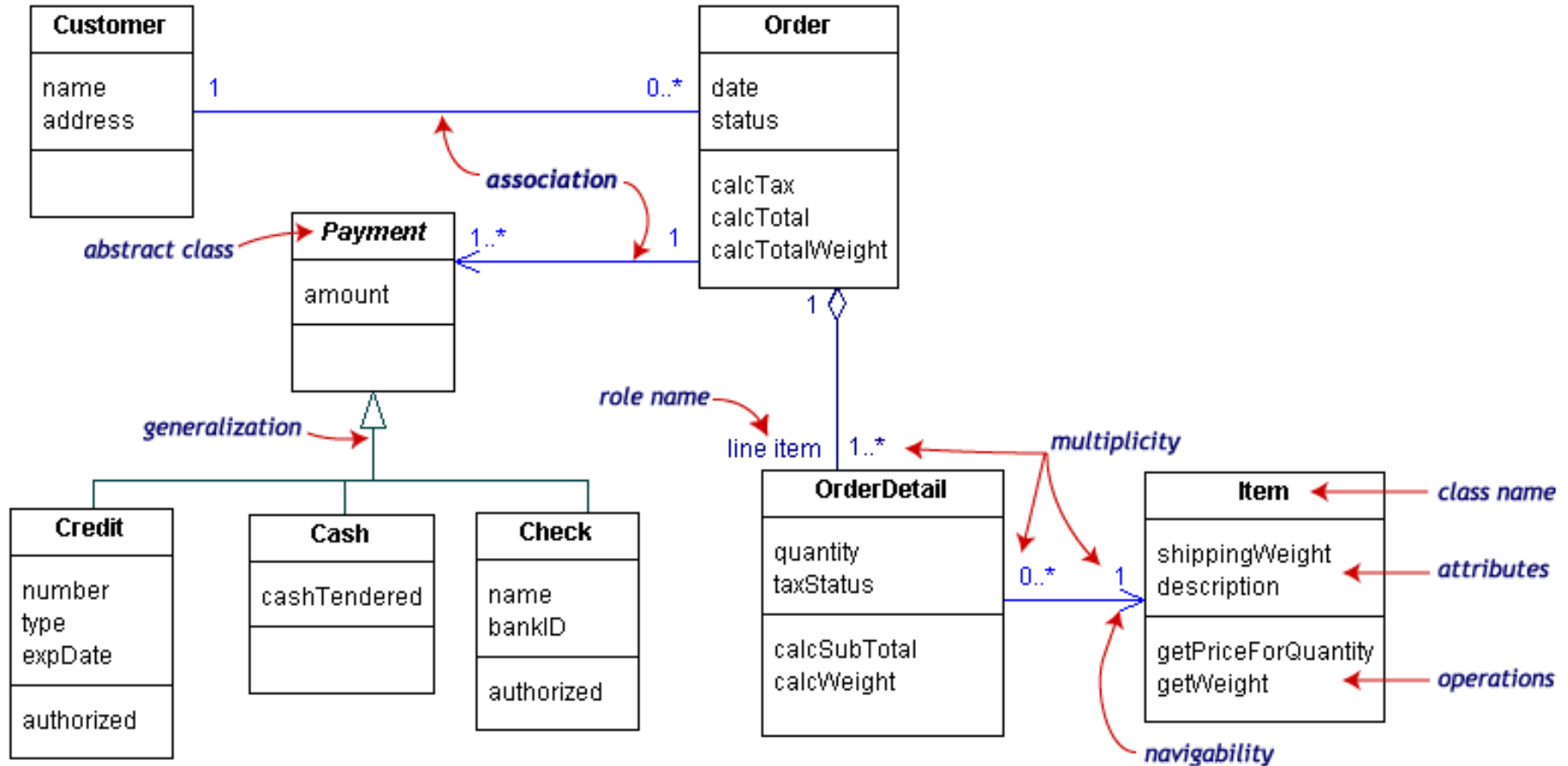


UML Multiplicities

Links on associations to specify more details about the relationship

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

UML Class Example



Object Diagrams

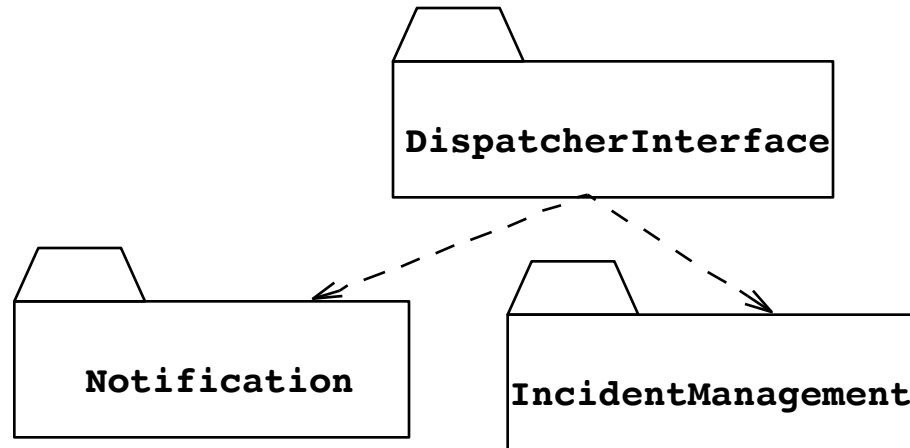
- Shows instances of Class Diagrams and links among them
- An object diagram is a snapshot of the objects in a system
 - At a point in time
 - With a selected focus
 - Interactions – Sequence diagram
 - Message passing – Collaboration diagram
 - Operation – Deployment diagram

Package Diagrams

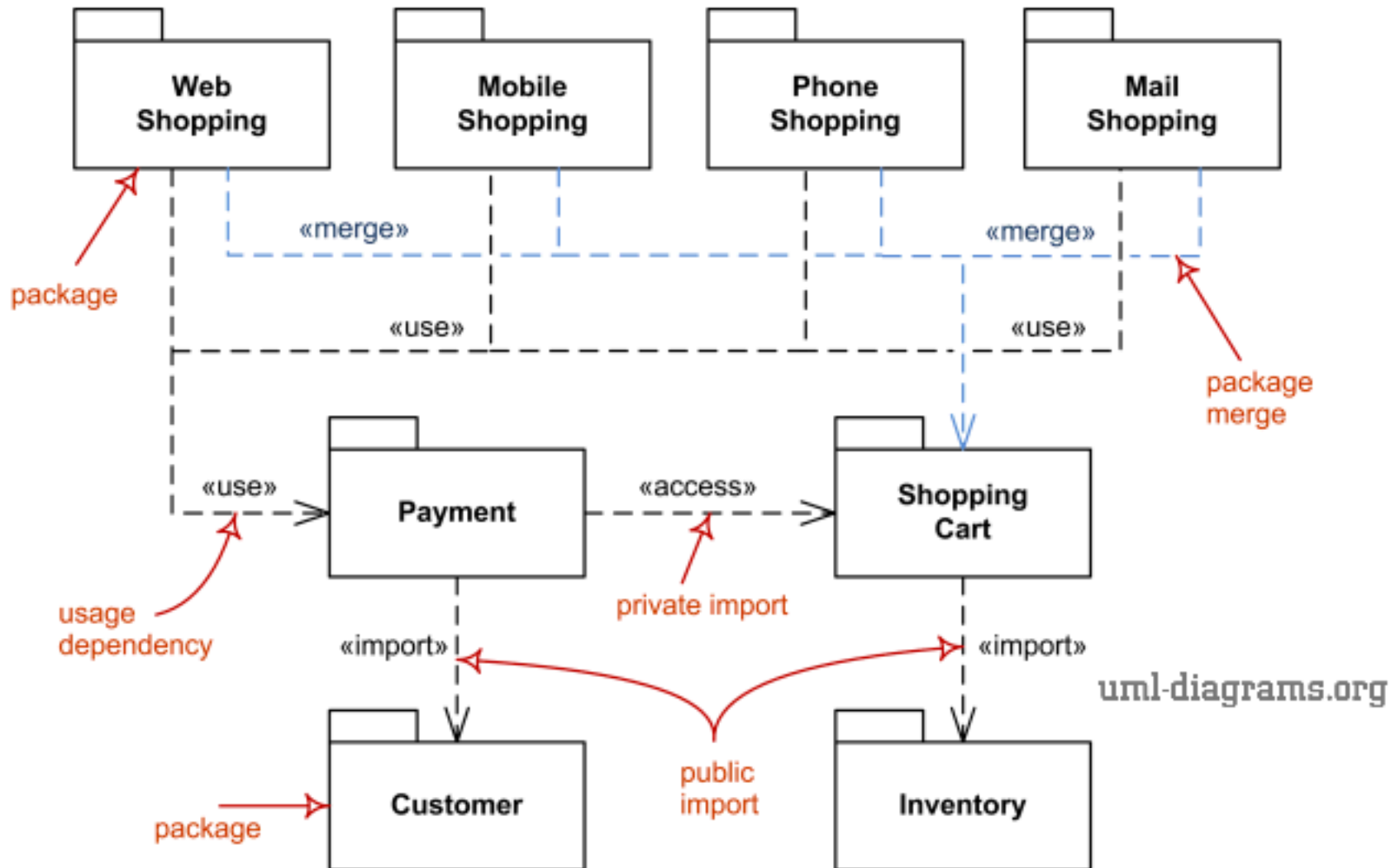
Package Diagrams

- 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.
 - Packages are the basic grouping construct with which you may organise UML models to increase their readability

Package Example



A Full Package Example



Interaction Diagrams

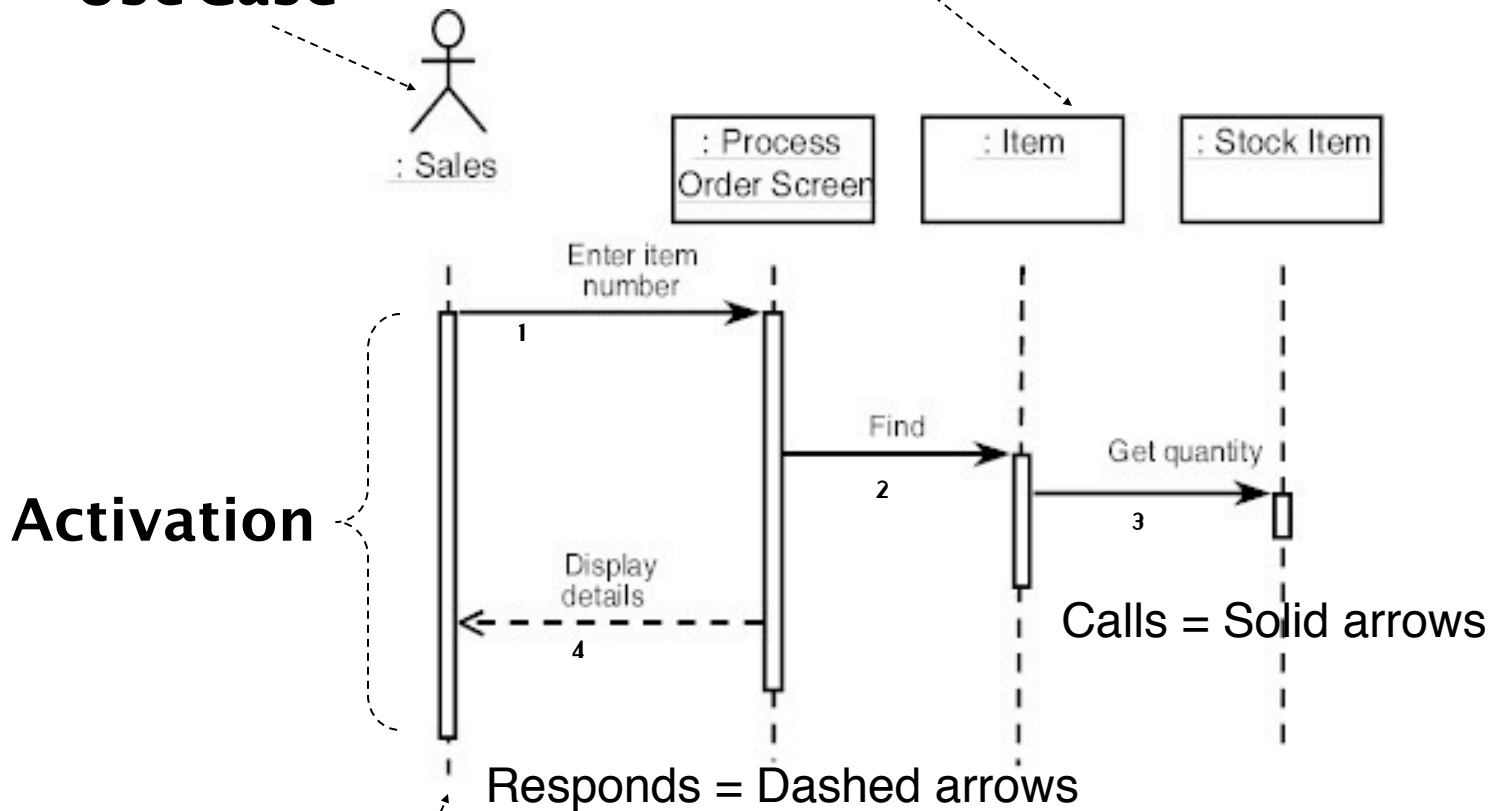
Interaction Diagrams

- Interaction diagrams are dynamic -- they describe how **objects** collaborate.
- Including but not limited to:
 - Sequence Diagram
 - Collaboration Diagram

Sequence Diagram

Actor from Use Case

Objects (Objects involved are listed left to right)

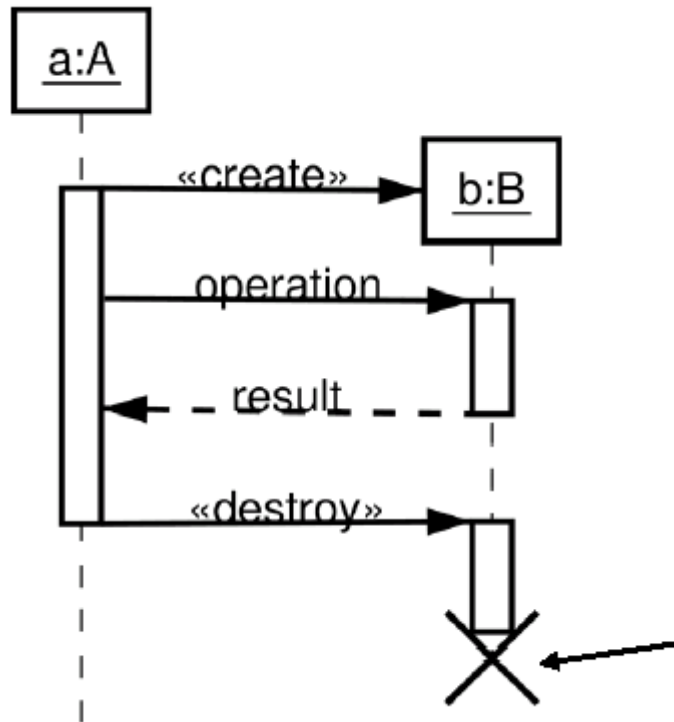


Activation

Lifeline

(Time progresses from top to bottom)

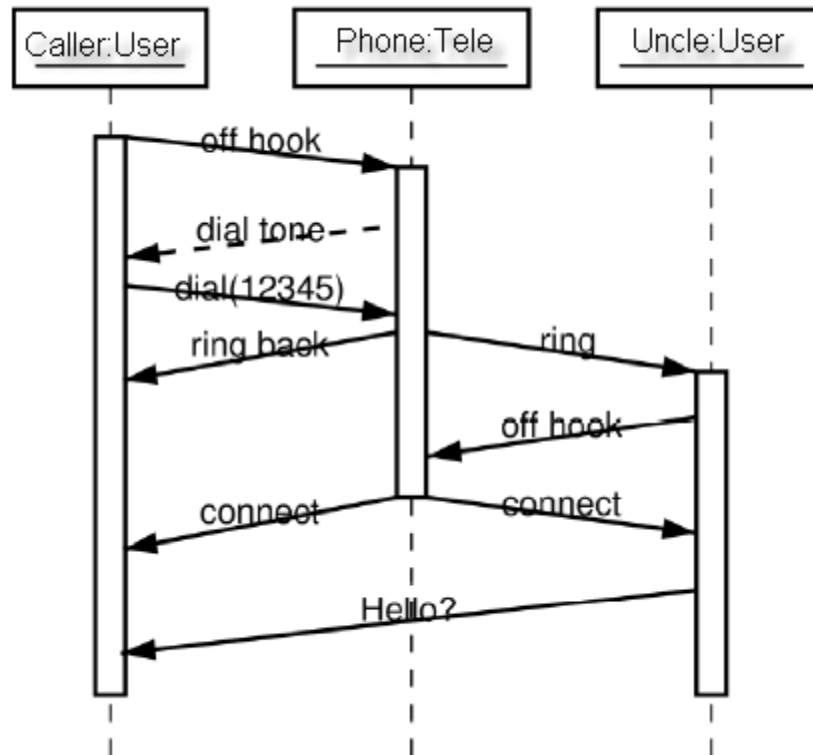
Sequence Diagram : Destruction



**Shows Destruction of b
(and Construction)**

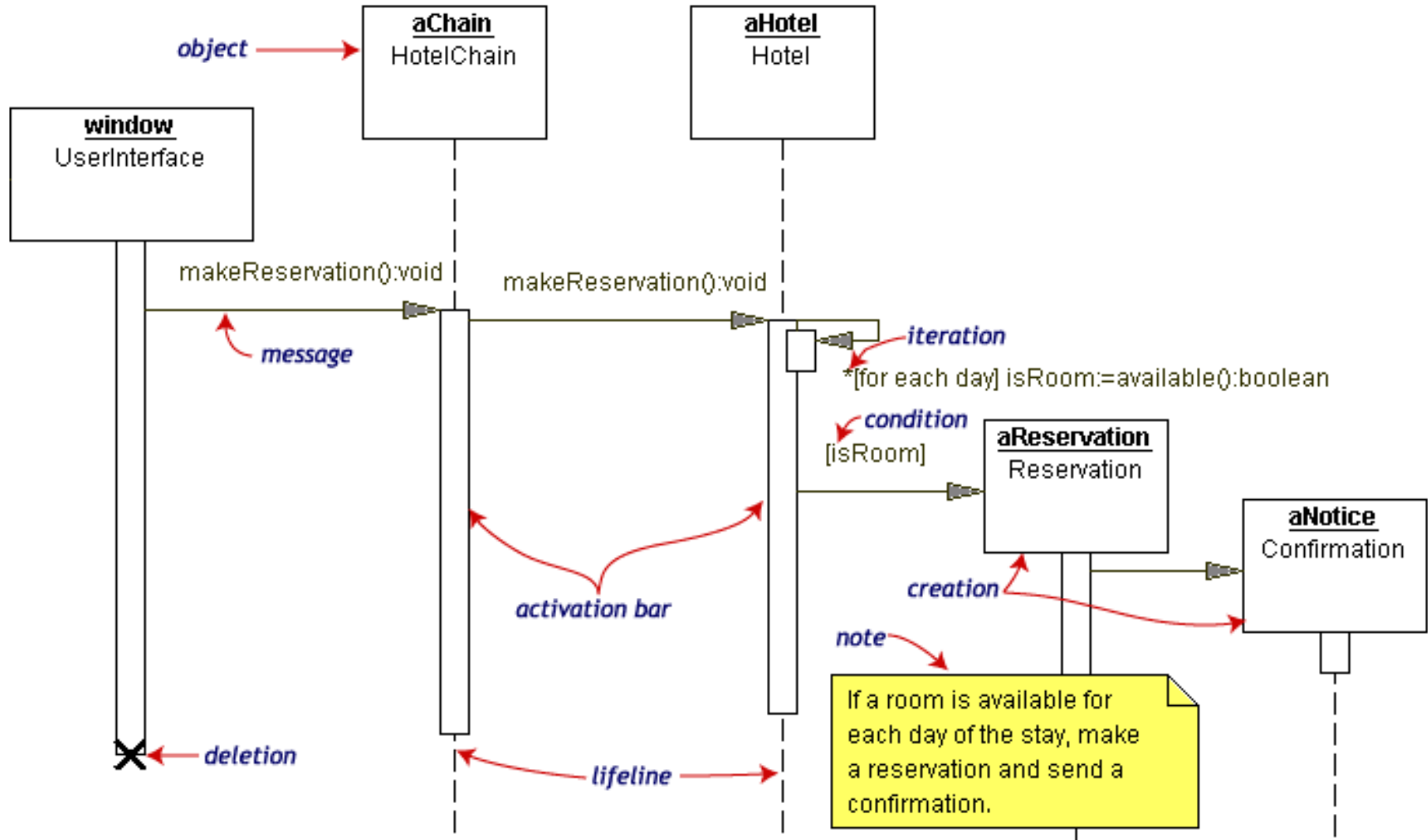
Sequence Diagram : Timing

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



Sequence Diagram Example

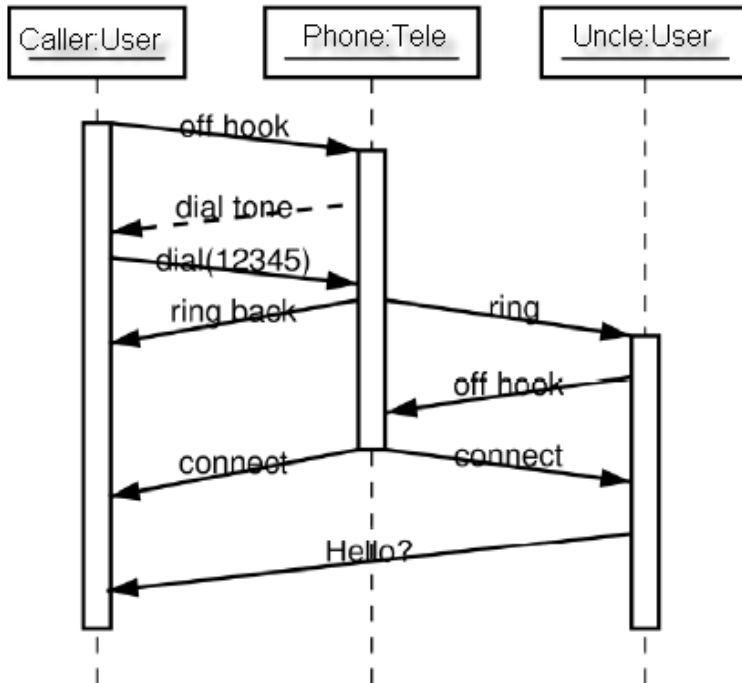
Hotel Reservation



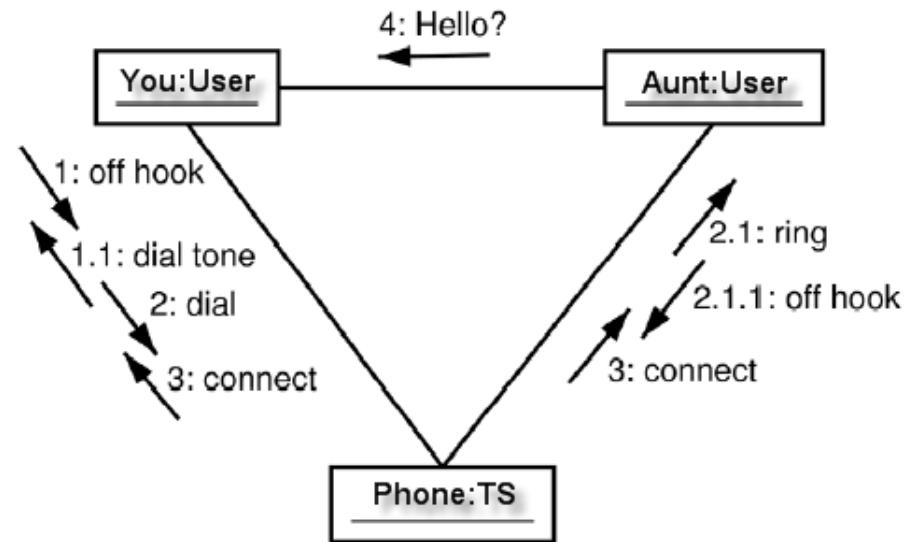
Collaboration Diagram

- Collaboration Diagrams show similar information to sequence diagrams, except that
 - The vertical sequence (lifeline) is missing
 - Object Links - solid lines between the objects that interact
 - On the links are Messages - arrows with one or more message name that show the direction and names of the messages sent between objects
- Emphasis on **how objects collaborate (interact)** as opposed to **lifetime** in the sequence diagram

Collaboration Diagram



Sequence Diagram



Collaboration Diagram

Activity Diagrams

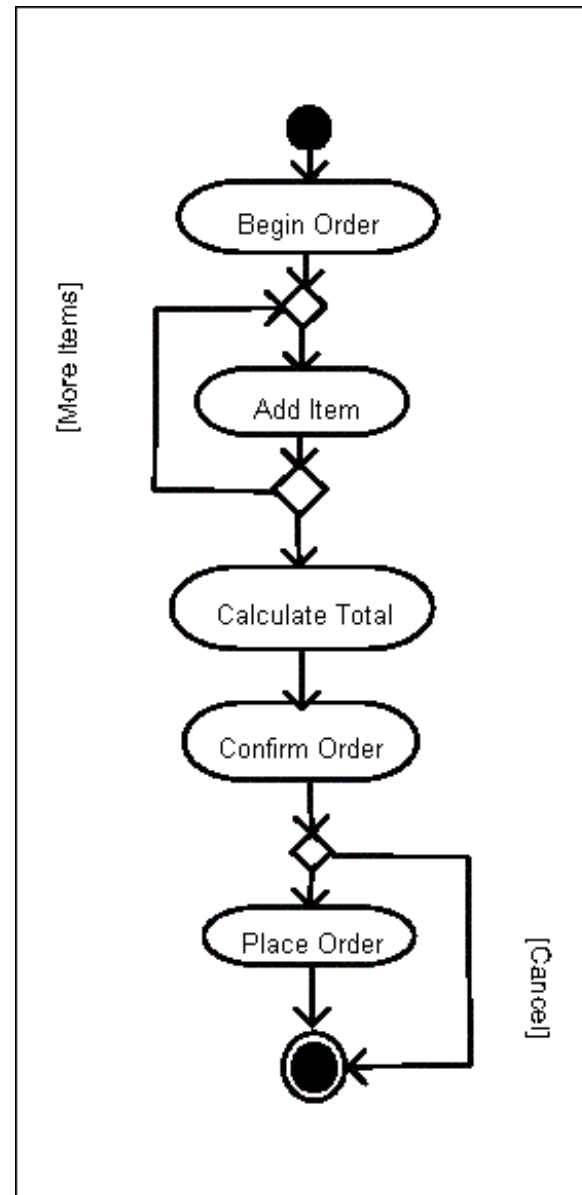
Activity Diagrams

- Flowchart: Displays the flow of activities involved in a single process (a.k.a. activity)
 - States
 - Describe what is being processed
 - Indicated by boxes with rounded corners
 - Swim lanes
 - Which object is responsible for what activity
 - Branch
 - Different branches of transitions
 - Indicated by a diamond
 - Fork
 - Concurrent (parallel) activities
 - Indicated by solid bars
 - Start and End

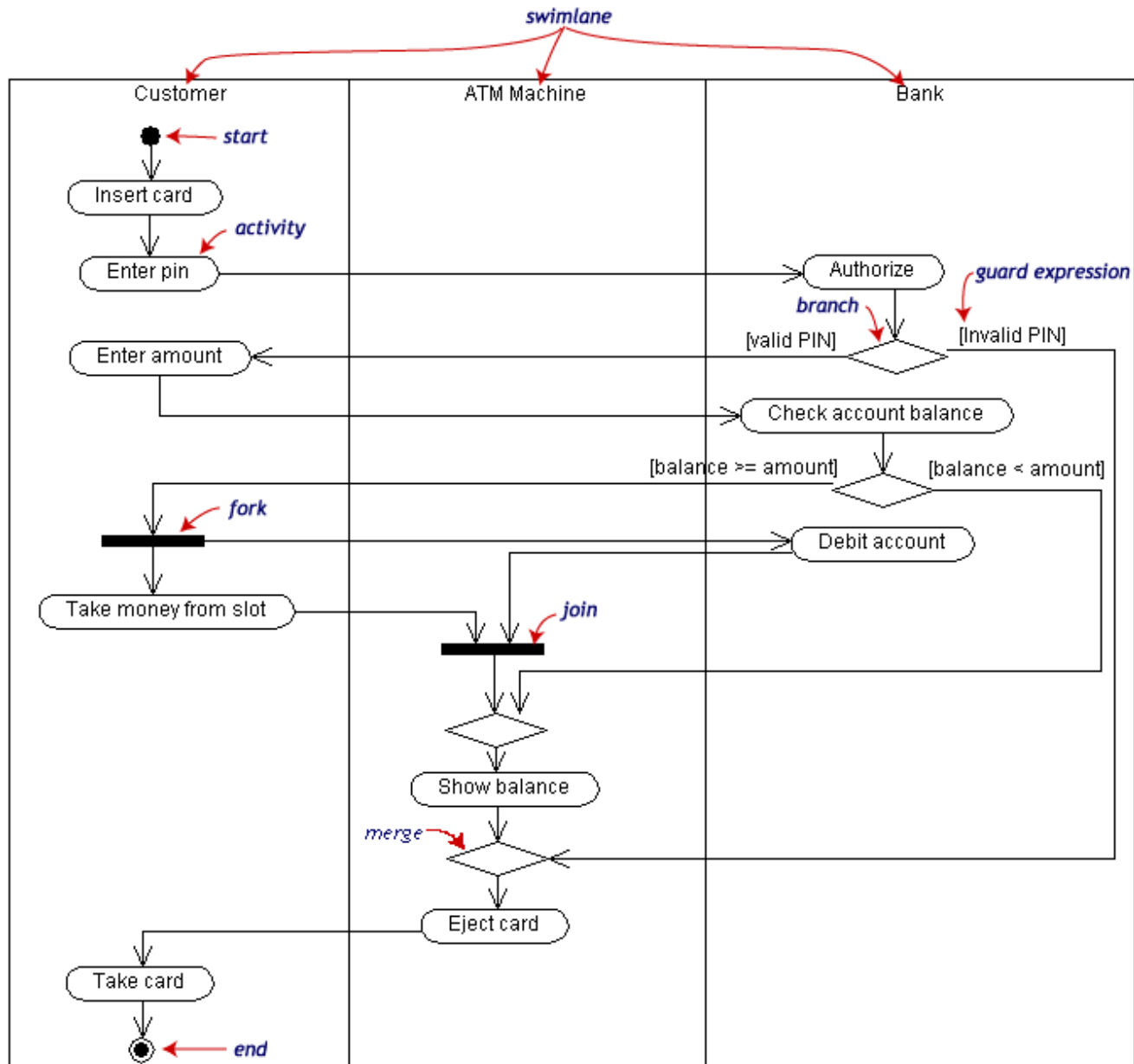


Sample Activity Diagram

- Ordering System
- May need multiple diagrams from other points of view



Activity Diagram Example



State Transition Diagrams

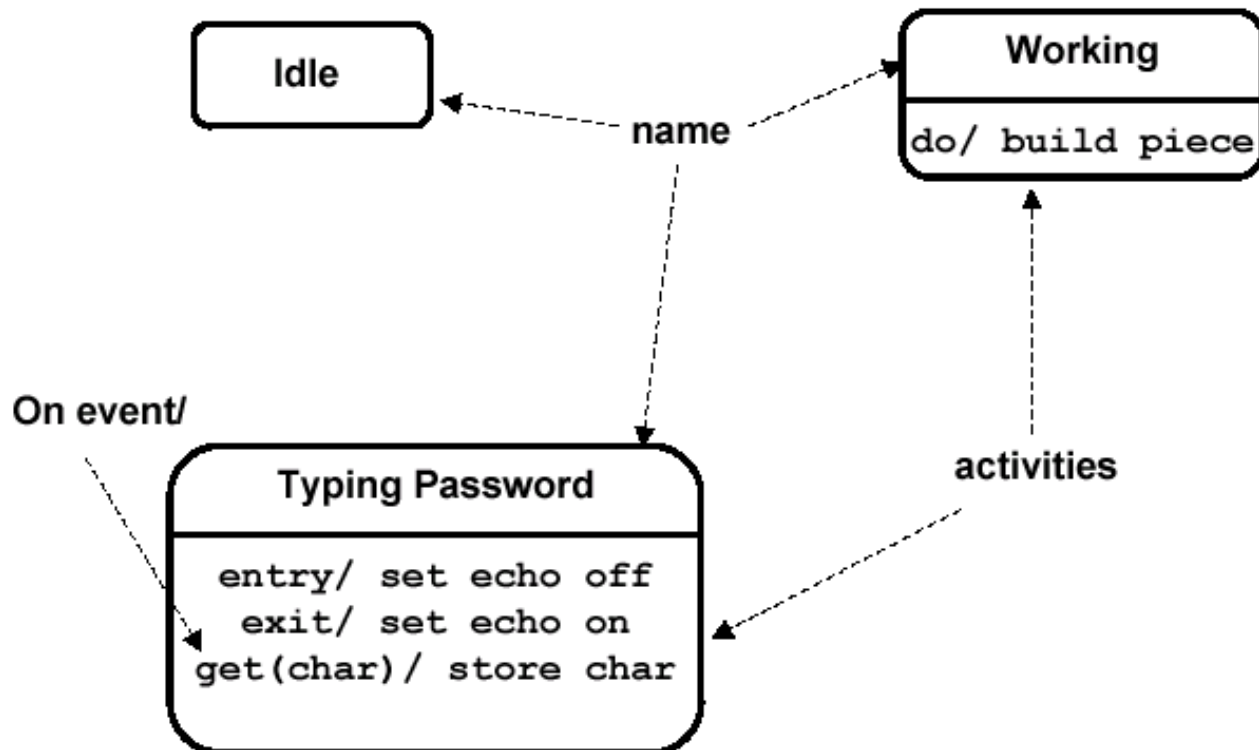
State Transition Diagrams

- States:
 - local attributes of an objects
 - Change in response to events (activities) or conditions
- Shows the possible states of the object and the transitions that cause a change in state
 - i.e. we receiving a call, one mark itself as on-call.
- Notation
 - States are rounded rectangles
 - Transitions are arrows from one state to another. Events or conditions that trigger transitions are written beside the arrows.
 - Initial and Final States indicated by circles as in the Activity Diagram
 - Final state terminates the action; may have multiple final states

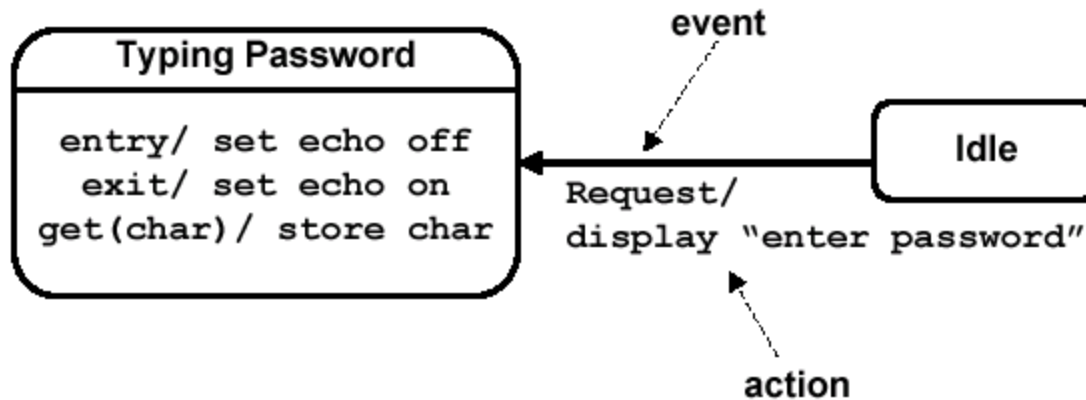
State Representation

- The set of properties and values describing the object in a well defined instant are characterised by
 - Name
 - Activities (executed inside the state)
 - Do/ activity
 - Actions (executed at state entry or exit)
 - Entry/ action
 - Exit/ action
 - Actions executed due to an event
 - Event [Condition] / Action ^Send Event

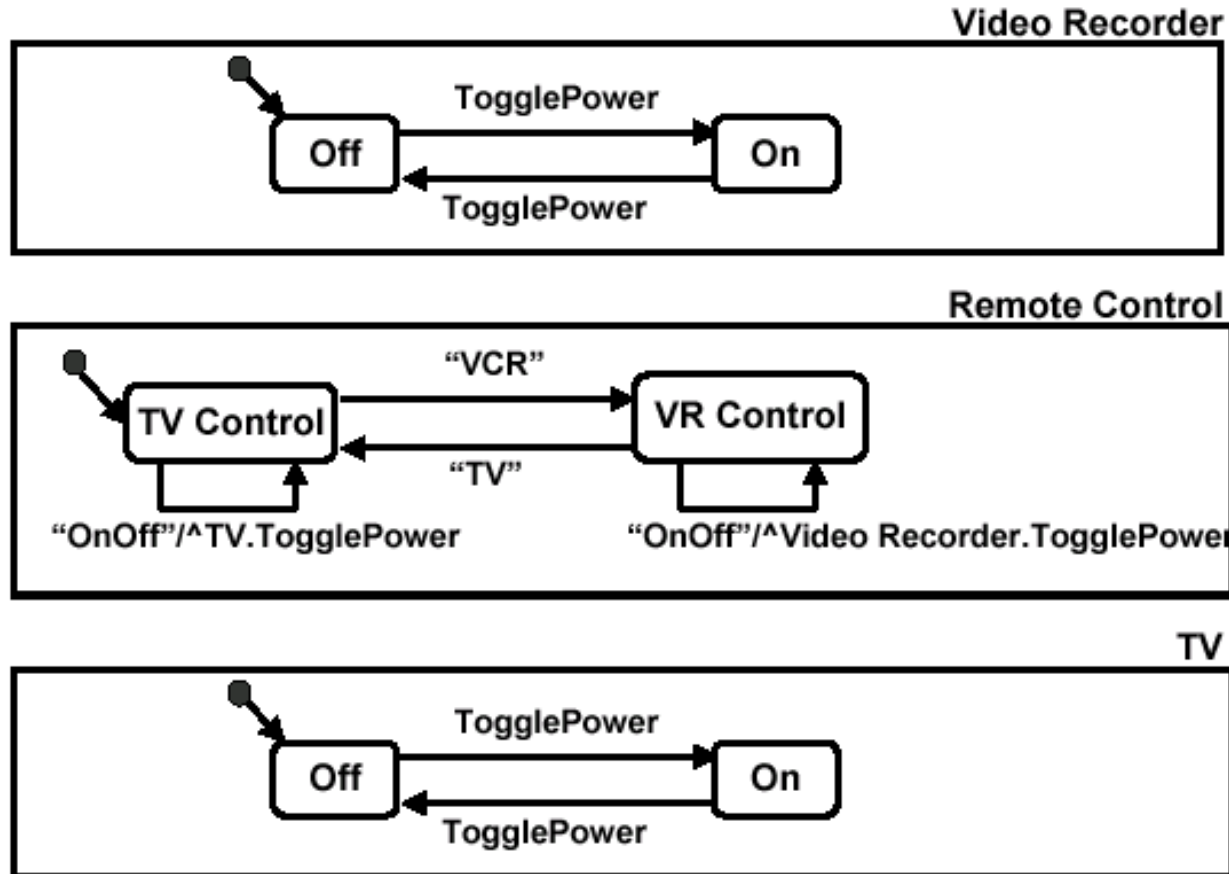
Notation for States



Simple Transition Example

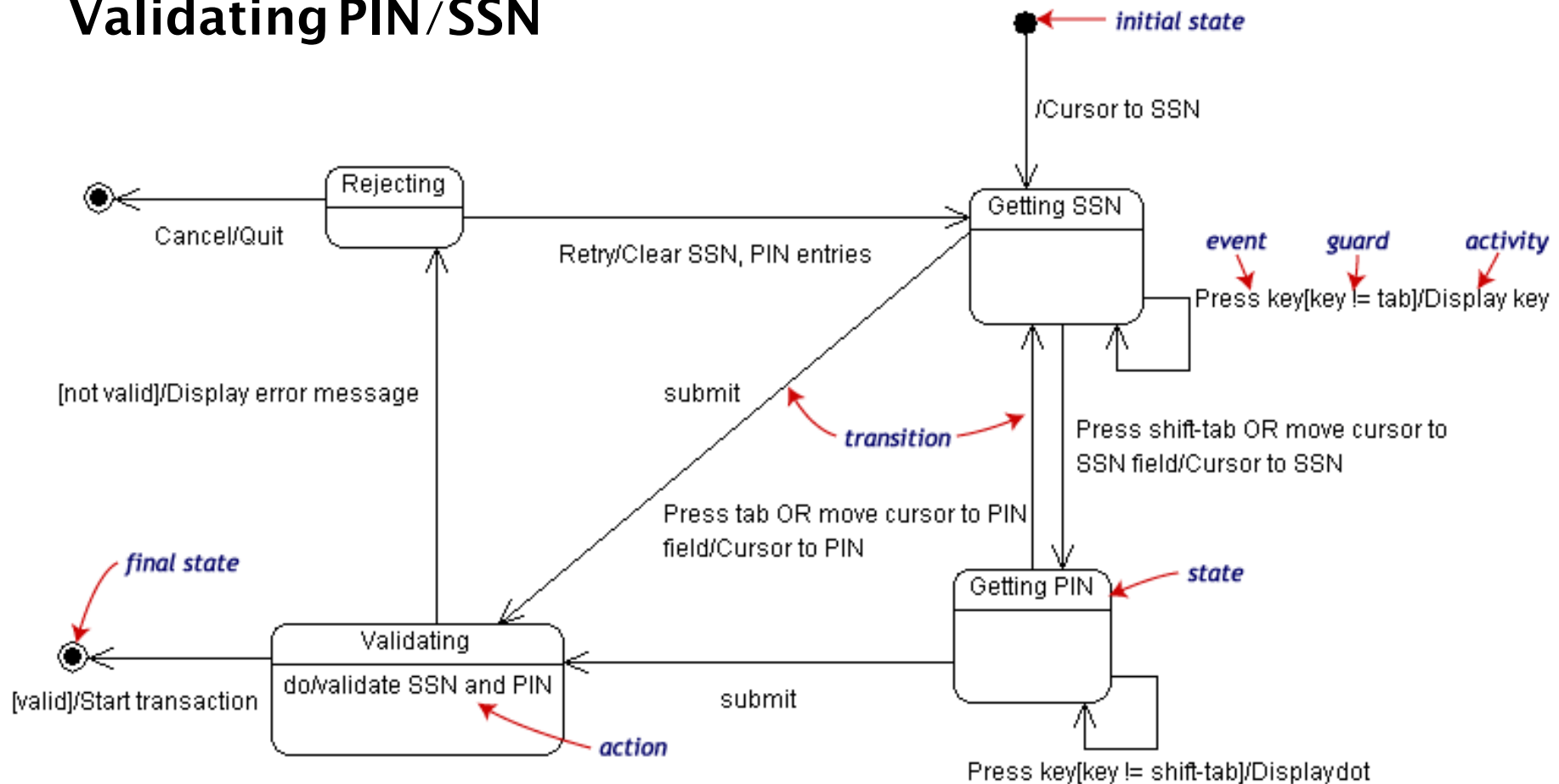


More Simple State Examples



State Transition Example

Validating PIN/SSN

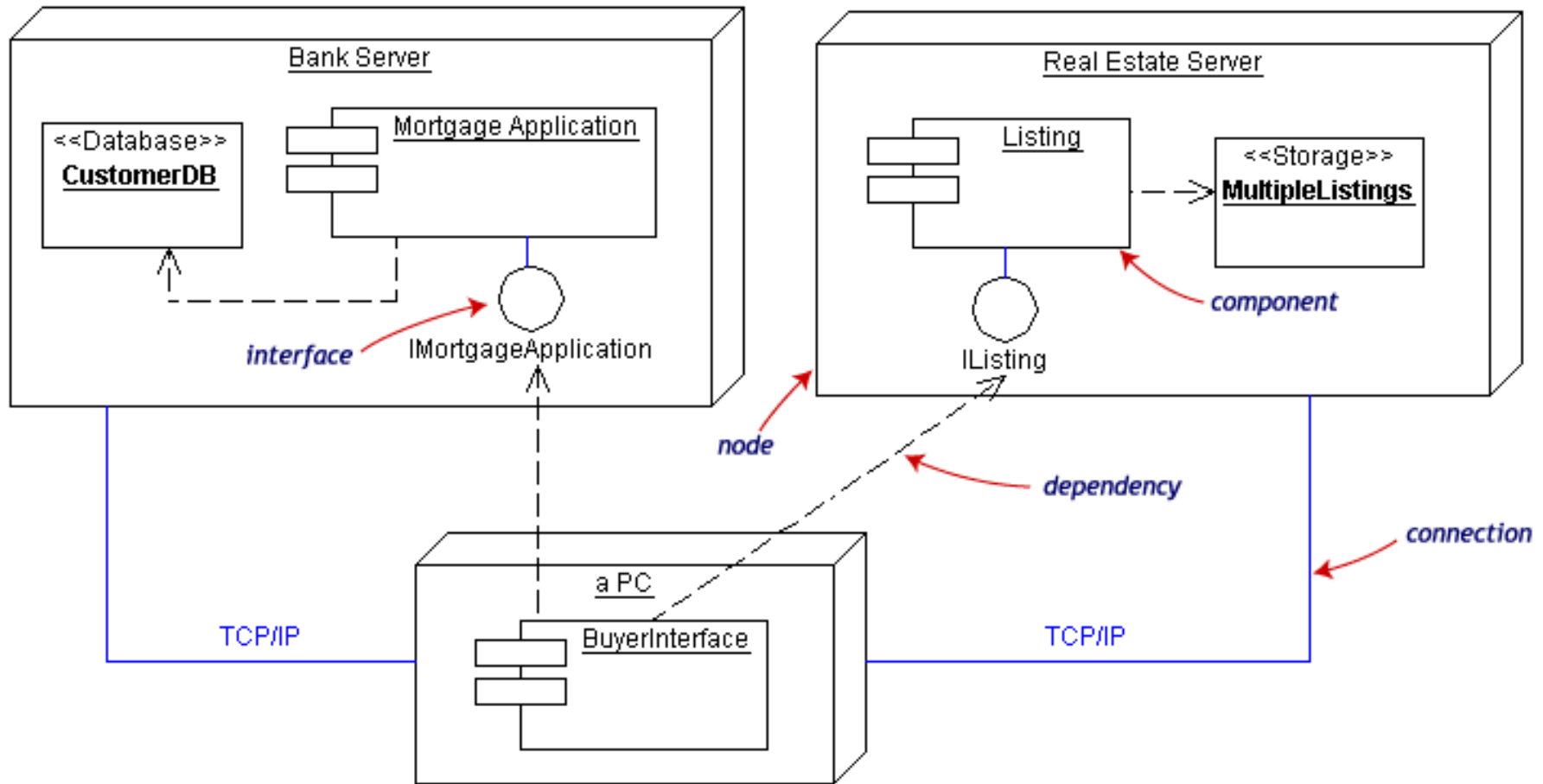


Deployment Diagrams

Deployment Diagrams

- Shows the physical architecture of the hardware and software of the deployed system
- Nodes
 - Typically contain packages (or components)
 - 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 Example



Summary and Tools

- UML is a modelling language that can be used independent of development
- Adopted by OMG and notation of choice for visual modelling
 - <http://www.omg.org/uml/>
- Creating and modifying UML diagrams can be labor and time intensive.
- Lots of tools exist to help
 - Repository (github) for a complete software development project
 - Software tools:
 - **Rational**, Cetus, Embarcadero
 - See <http://plg.uwaterloo.ca/~migod/uml.html> for a list of tools, some free