소프트웨어공학 원리 (SEP521)

Introduction to UML

Jongmoon Baik

Design using UML 2.0

Contents

- Why model
- What is UML
- UML history
- UML 2.0
- Diagram/View paradigm
- UML diagrams
  - Use case
  - Class
  - Sequence
  - State machine

Here, we focus on..

Analyst

Tau G2

Designer

Requirement Analysis Technique

Software Architecture/ Design Technique

Analysis

Design

Analyst

Telelogic DOORS

Requirement Elicitation Technique

Conceptualization

Programmer

Eclipse, Jbuilder

Programming Technique

Code

Software Configuration Management

SW Development Process

Software Requirement Management

Maintenance

Test

Process

Technology

People

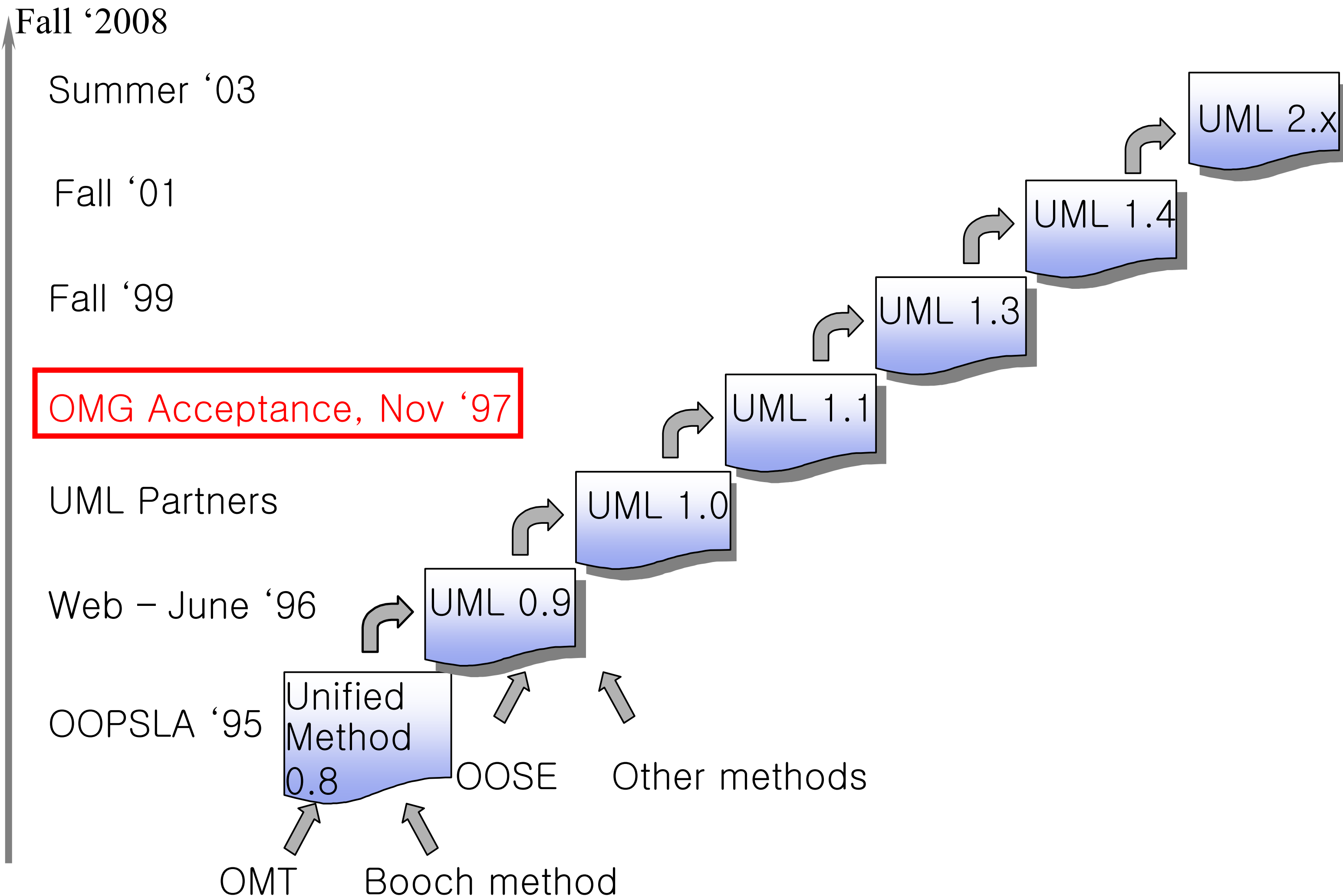
Why model

- To easily communicate information between different stakeholders in an unambiguous way
- To specify target-language-independent designs
- To provide structure for problem solving
- To provide mechanisms(abstractions, views, filtering, structure) to manage complexity
- To be able to easily experiment to explore multiple solutions

What is UML?

- **U**nified **M**odeling **L**anguage
  - Visual language for specifying, constructing and documenting
- Maintained by the OMG (Object Management Group)
  - Website: http://www.omg.org
- Object-oriented
- Model / view paradigm
- Target language independent

UML history

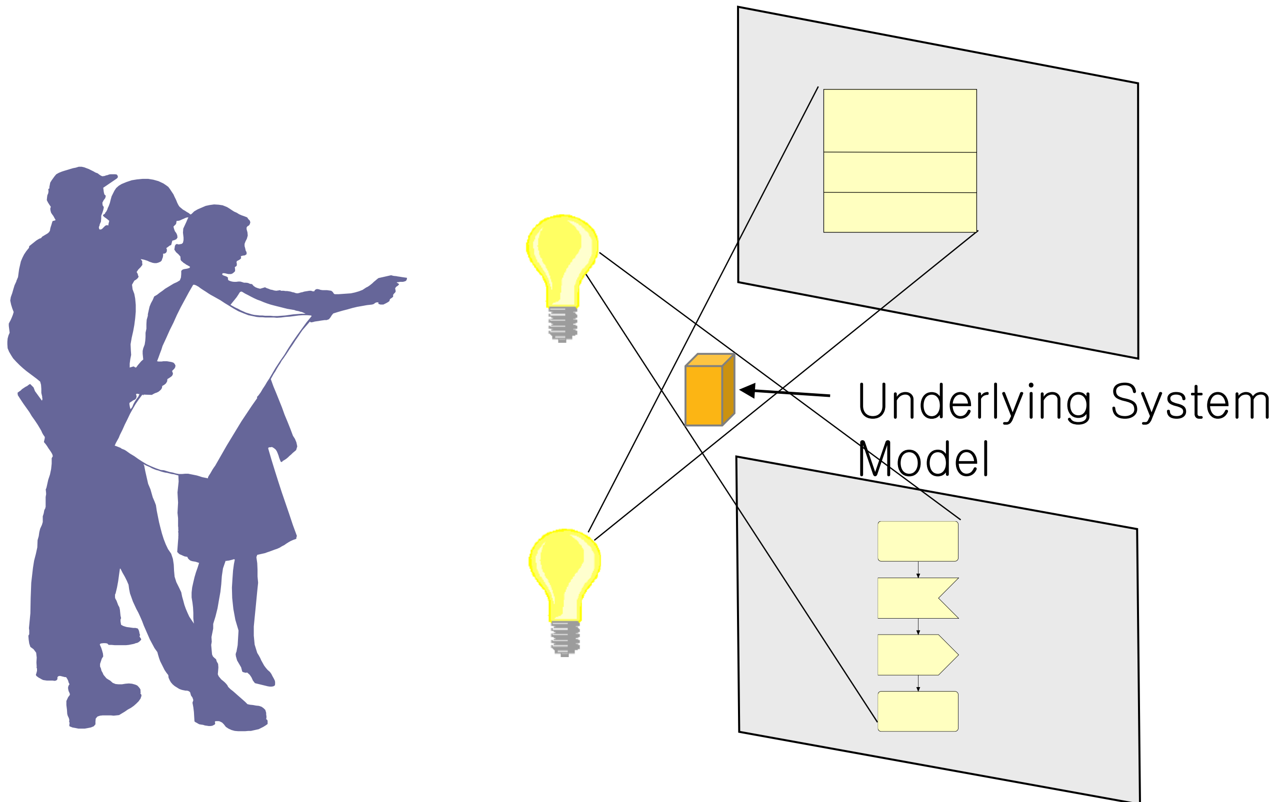


UML 2.0

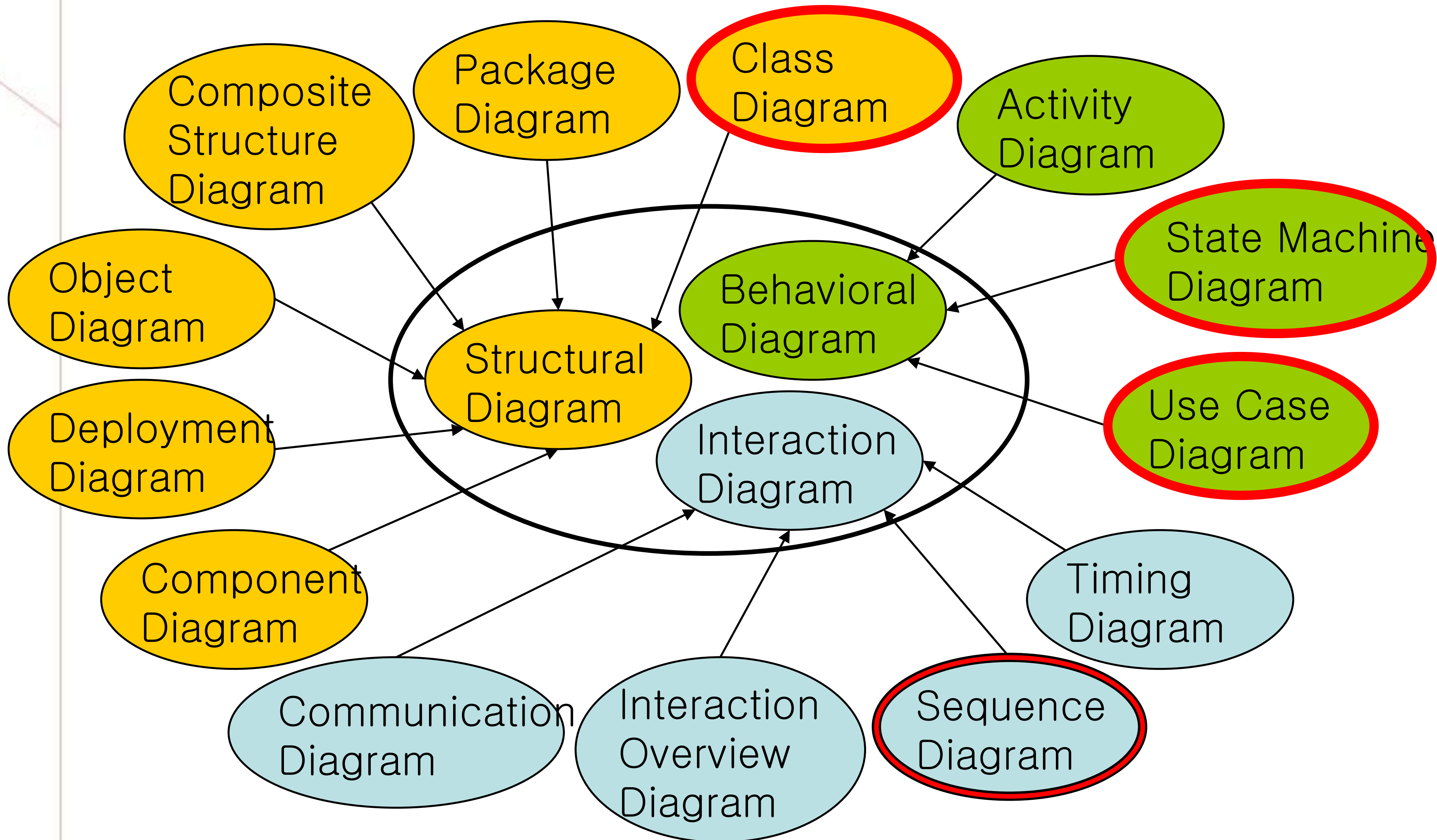
- UML 2.0 leverages the industry's investment in UML 1.x and makes UML comprehensive, scalable and mature
- UML 2.0 designed to solve the key UML 1.x issues
- Major improvements in UML 2.0 include:
  - New internal structure diagrams support precise definition of architecture, interfaces and components
  - Improved scalability in state machine and sequence diagrams
  - Better semantic foundation enables advanced model verification and full code generation

Diagram/view paradigm

- Each diagram is just a view of part of the system
- Together, all diagrams provides a complete picture

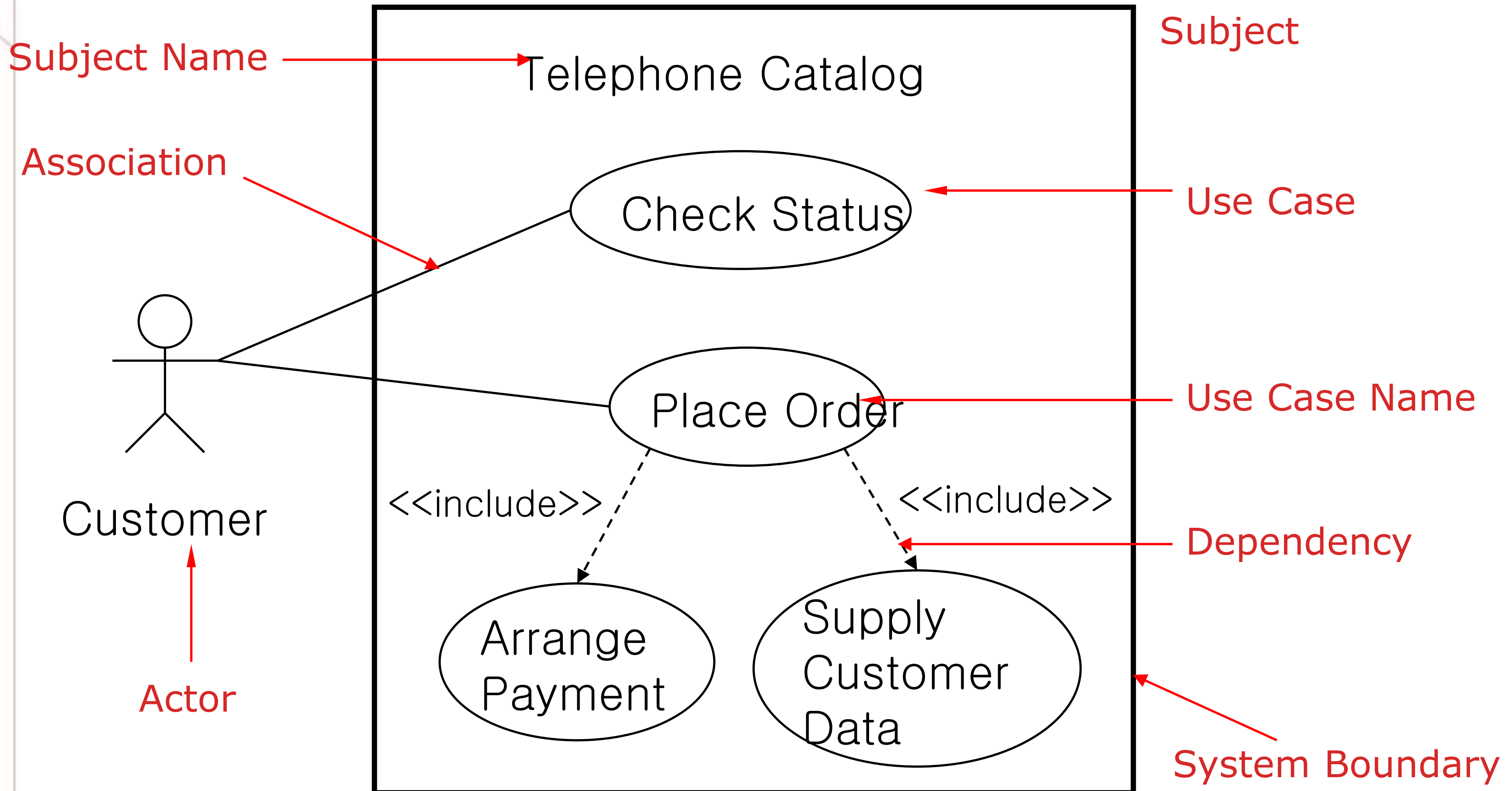


UML diagrams



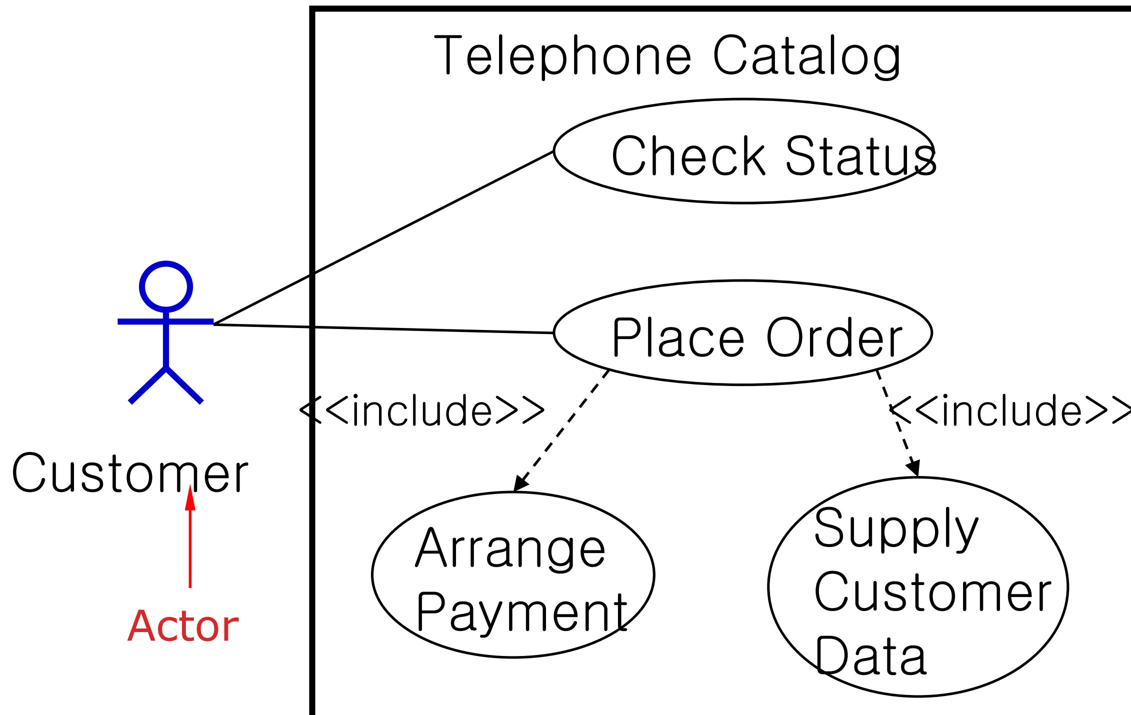
Use Case Diagram

- Describe WHAT the system will do at a high-level



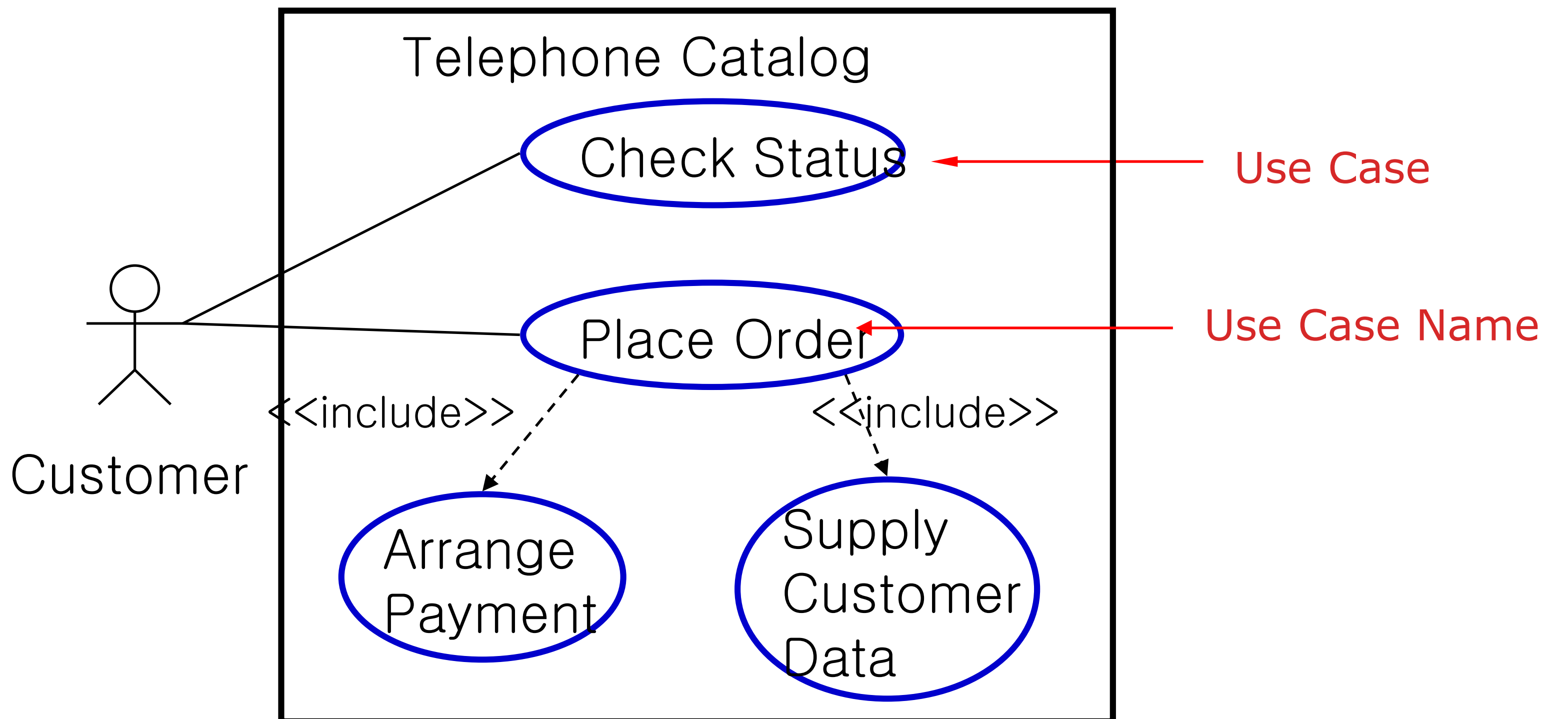
Actor

- Someone or some thing that must interact with the system under development
  - Users, external systems, devices



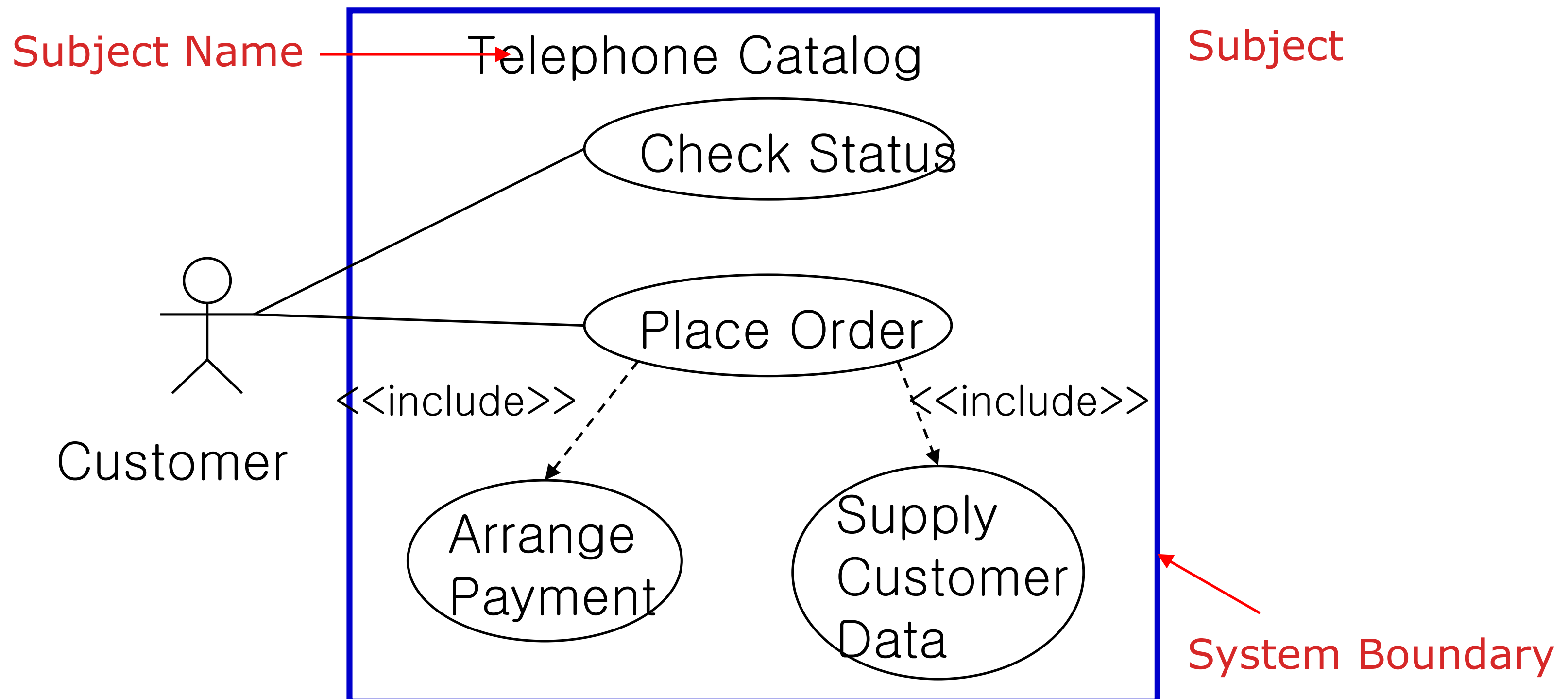
Use Case

- Functionality that the system shall offer to an actor
- Interaction between one or more actors and the system



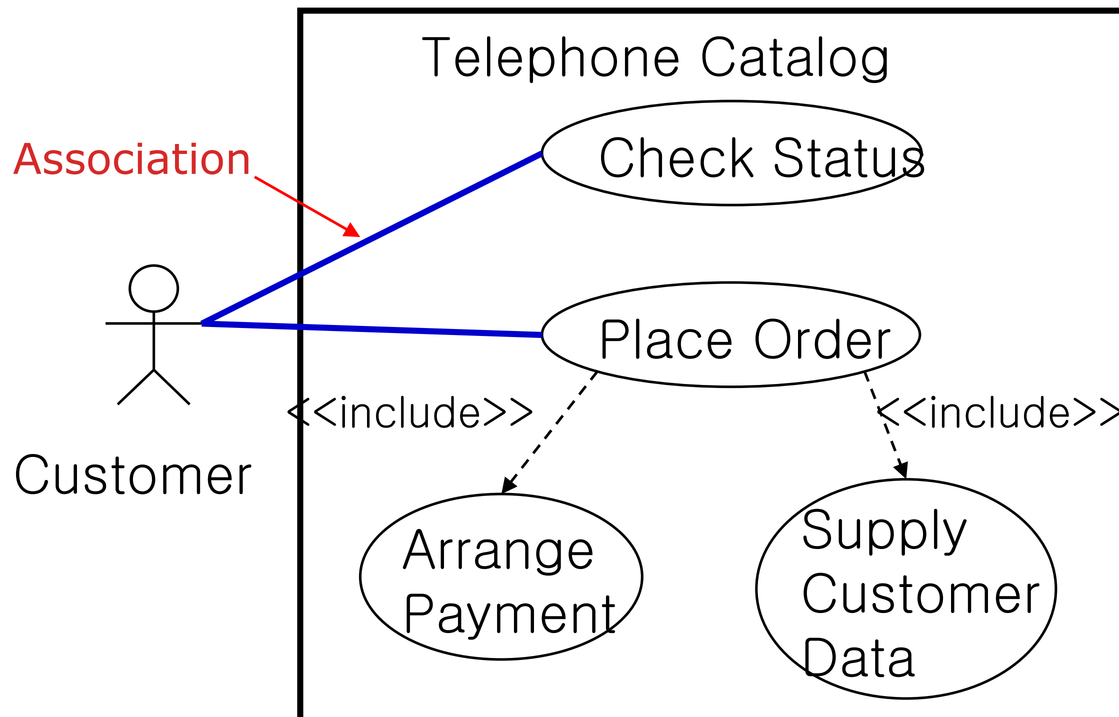
Subject Symbol

- Indicate system boundary
- Represent the system begin developed
  - All actors who interact with the system are outside of it



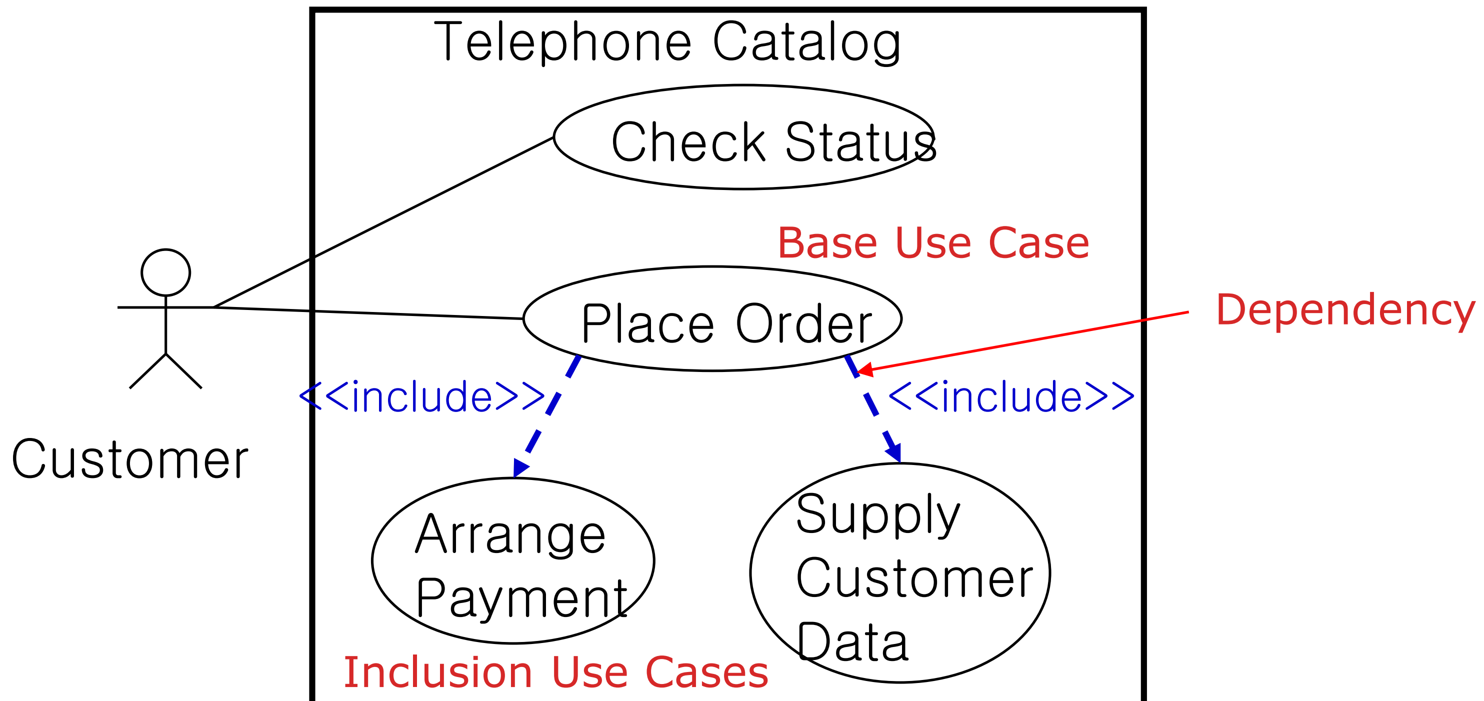
Association

- Drawn between an actor and a use case
- Represent bi-directional communication between the actor and the system



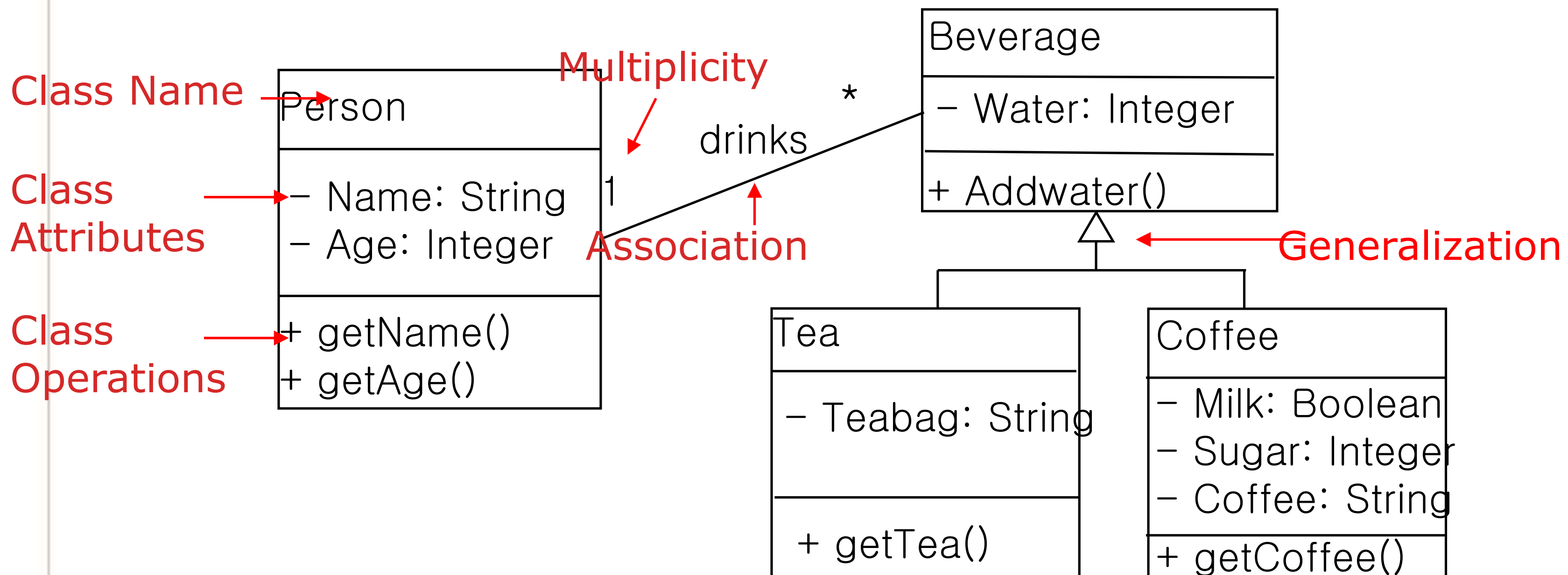
Dependency – Include

- Represent relationship from a *base* to an *inclusion* use case
- Imply a Use Case calls another Use Case
- Primarily used to reuse behavior common to several Use Cases



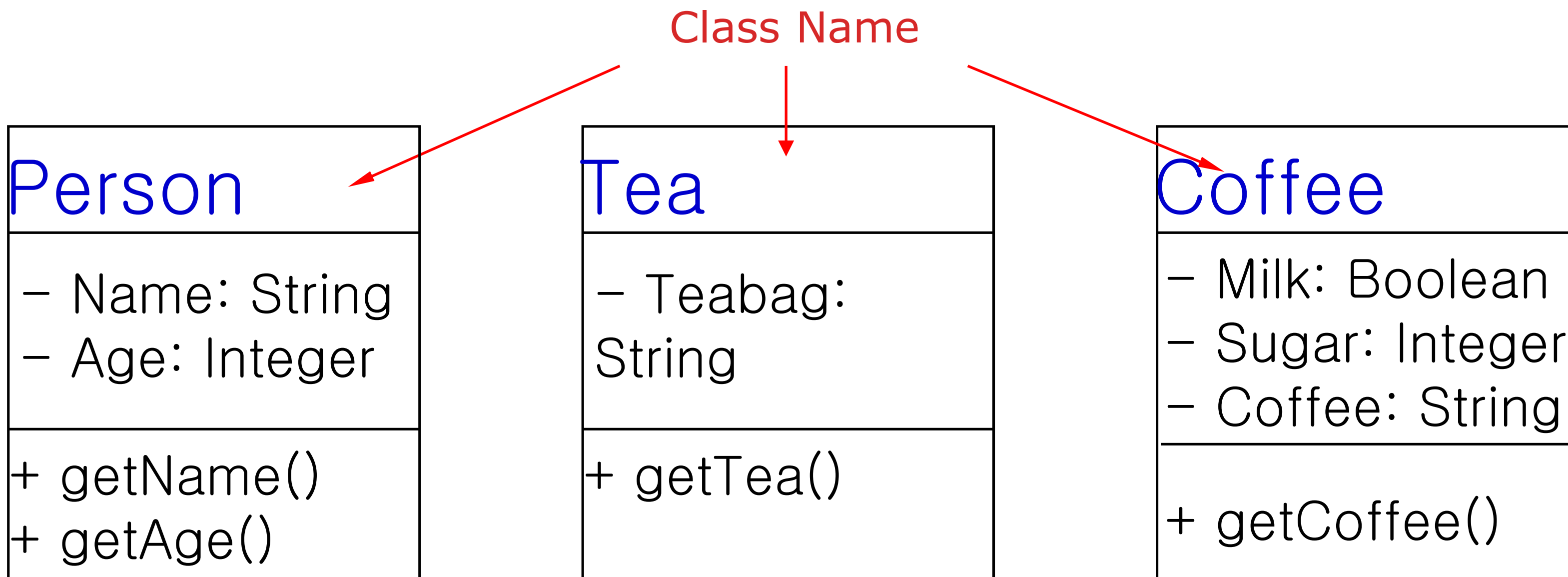
Class Diagram

- Description of static structure
  - Showing the types of objects in a system and the relationships between them
- Foundation for the other diagrams



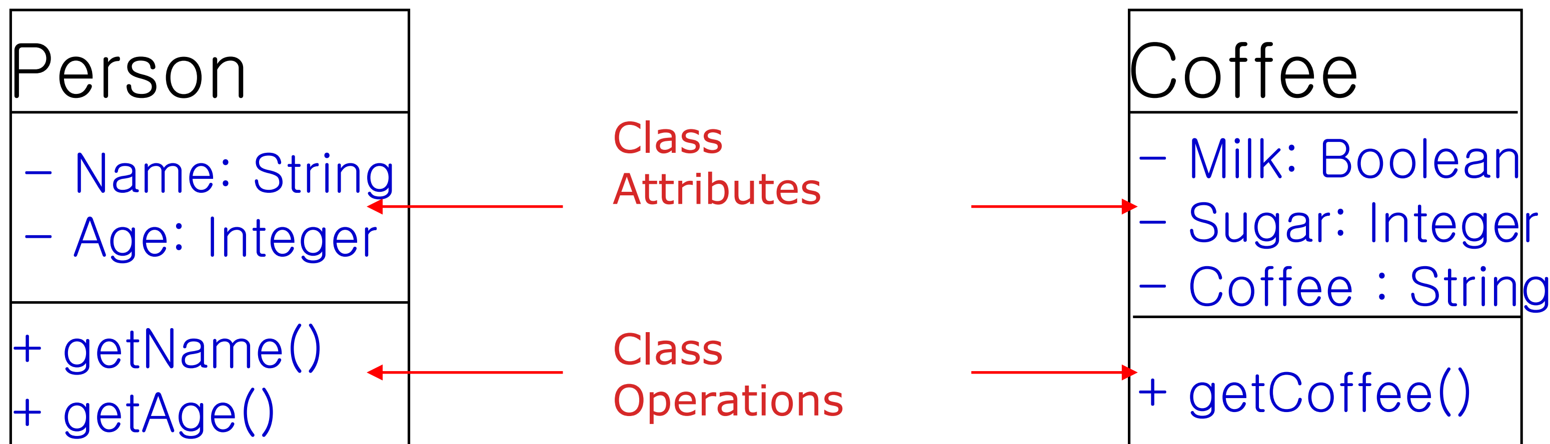
Classes

- Most important building block of any object-oriented system
- Description of a set of objects
- Abstraction of the entities
  - Existing in the problem/solution domain



Attributes and Operations

- Attributes
  - Represent some property of the thing being modeled
  - Syntax: attributeName : Type
- Operations
  - Implement of a service requested from any object of the class
  - Syntax: operationName(param1:type, param2:type, …) : Result

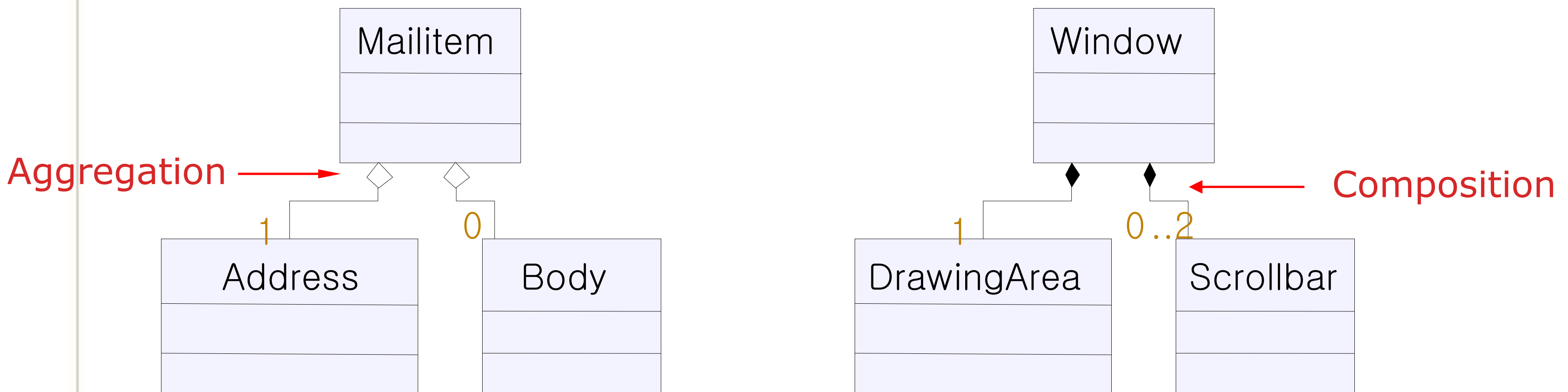


Association and Multiplicity

- Association
  - Relationship between classes that specifies connections among their instances
- Multiplicity
  - Number of instances of one class related to ONE instance of the other class

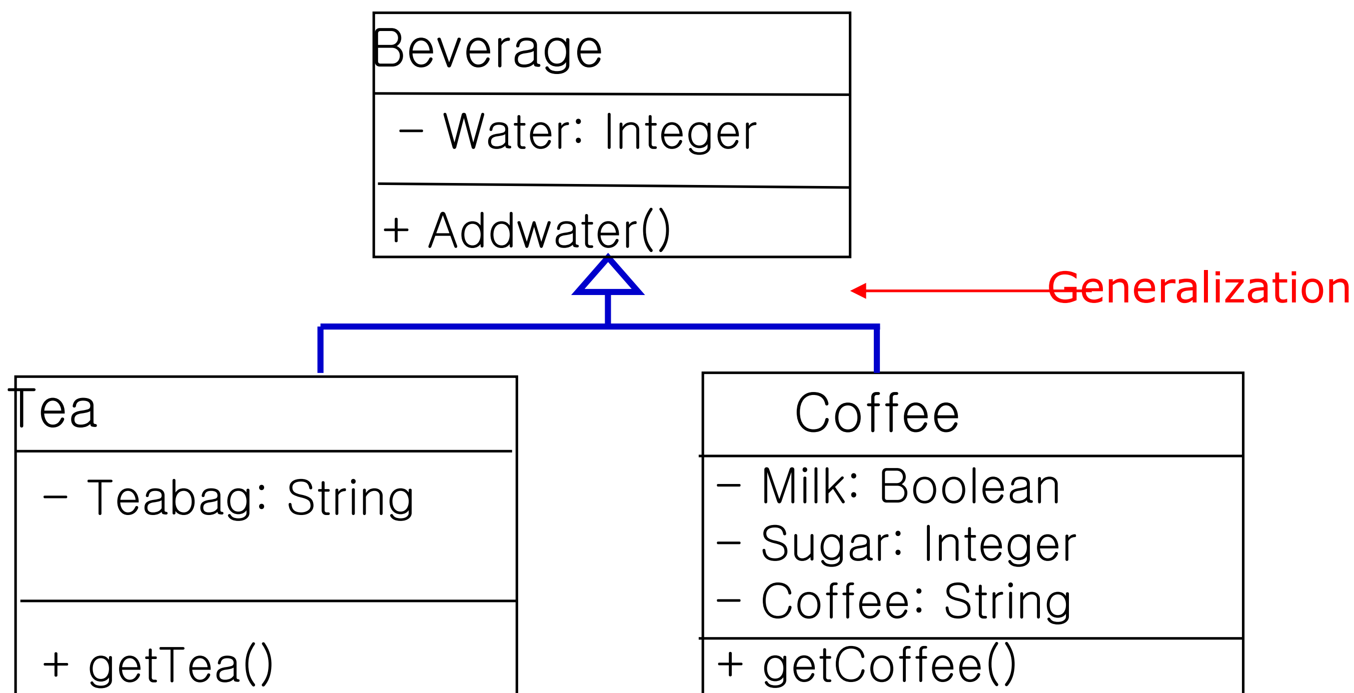
Aggregations and Compositions

- Aggregation
  - Weak "whole-part" relationship
    - Mailitem 'has a' address
- Composition
  - Strong "whole-part" relationship between elements
    - Window 'contains a' scrollbar



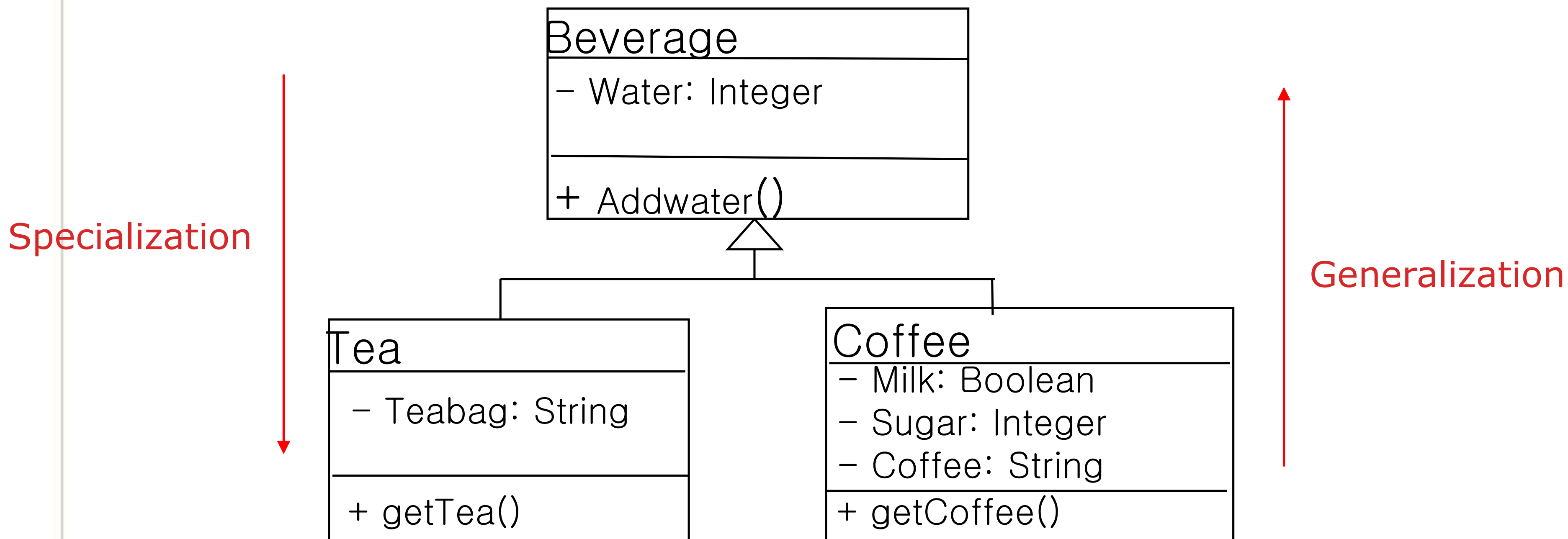
Inheritance

- Relationship between superclass and subclasses
  - All attributes and operations of the superclass are part of the subclasses



Generalization/Specialization

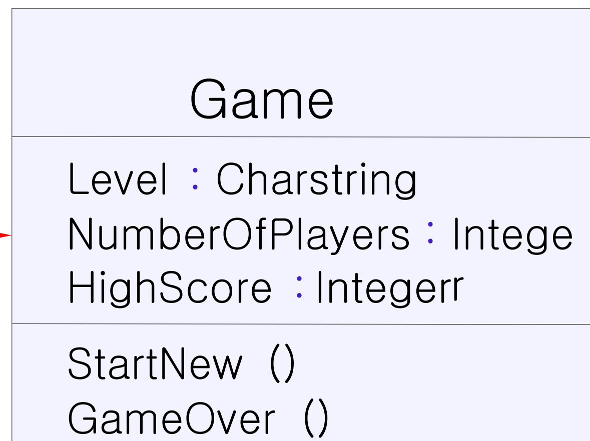
- Generalization
  - Building a more general class from a set of specific classes
- Specialization
  - Creating specialized classes base on a more general class



Active vs. Passive Class

- Active class
  - Own a thread control and can initiate control activity
    - Used when asynchronous communication is necessary
    - Typically modeled with a statemachine of its behavior
    - Encapsulated with ports and interfaces
- Passive class
  - Created as part of an action by another object
    - Own address space, but not thread of control

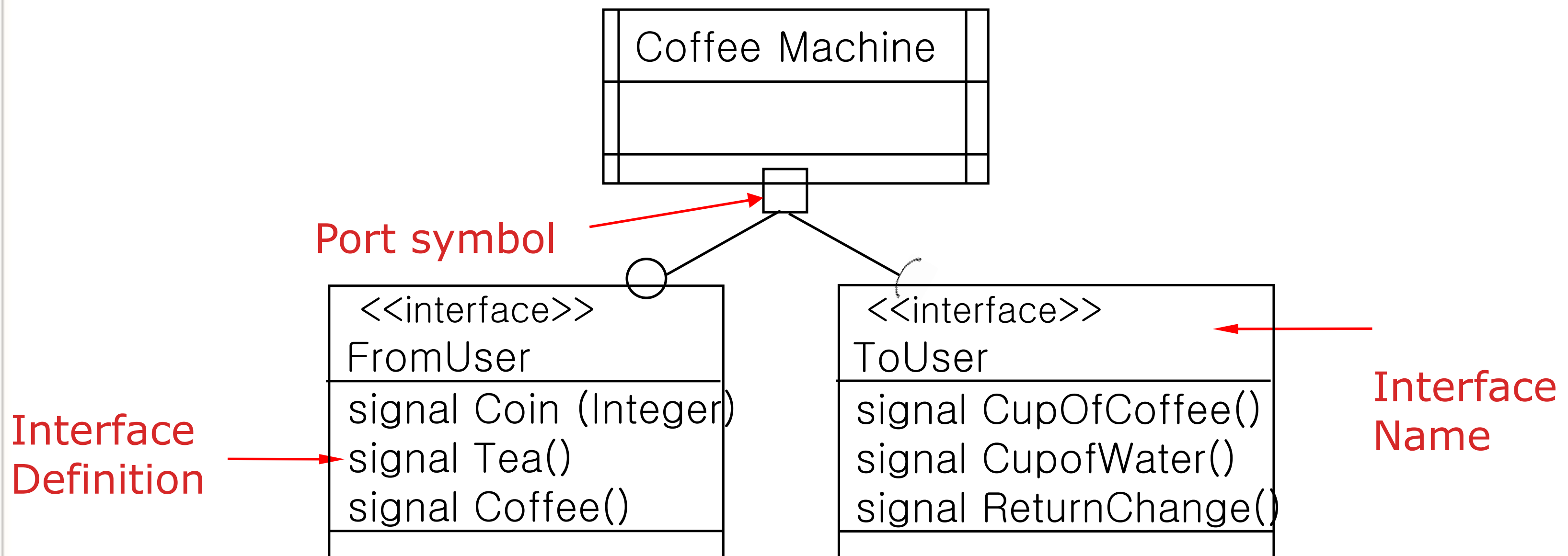
Passive class



Active class

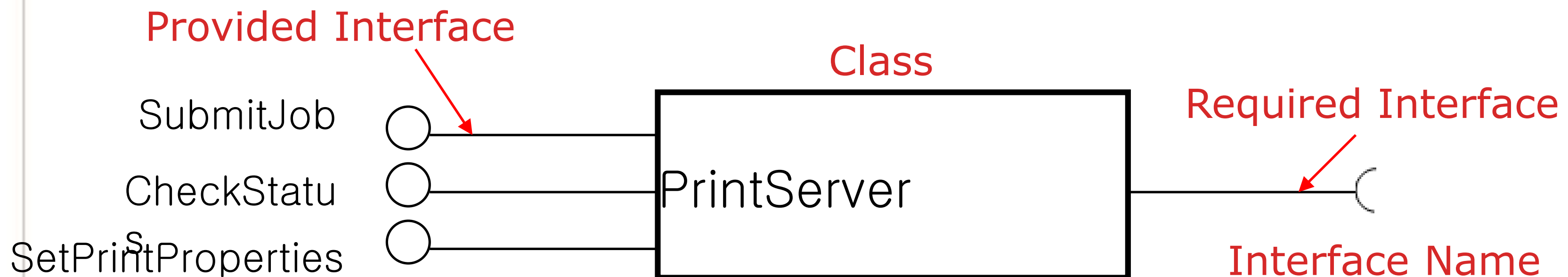
Ports and Interfaces

- Ports
  - Define an interaction point on a classifier
- Interfaces
  - Declaration of a coherent set of public features and obligations
    - Contract between providers and consumers of services

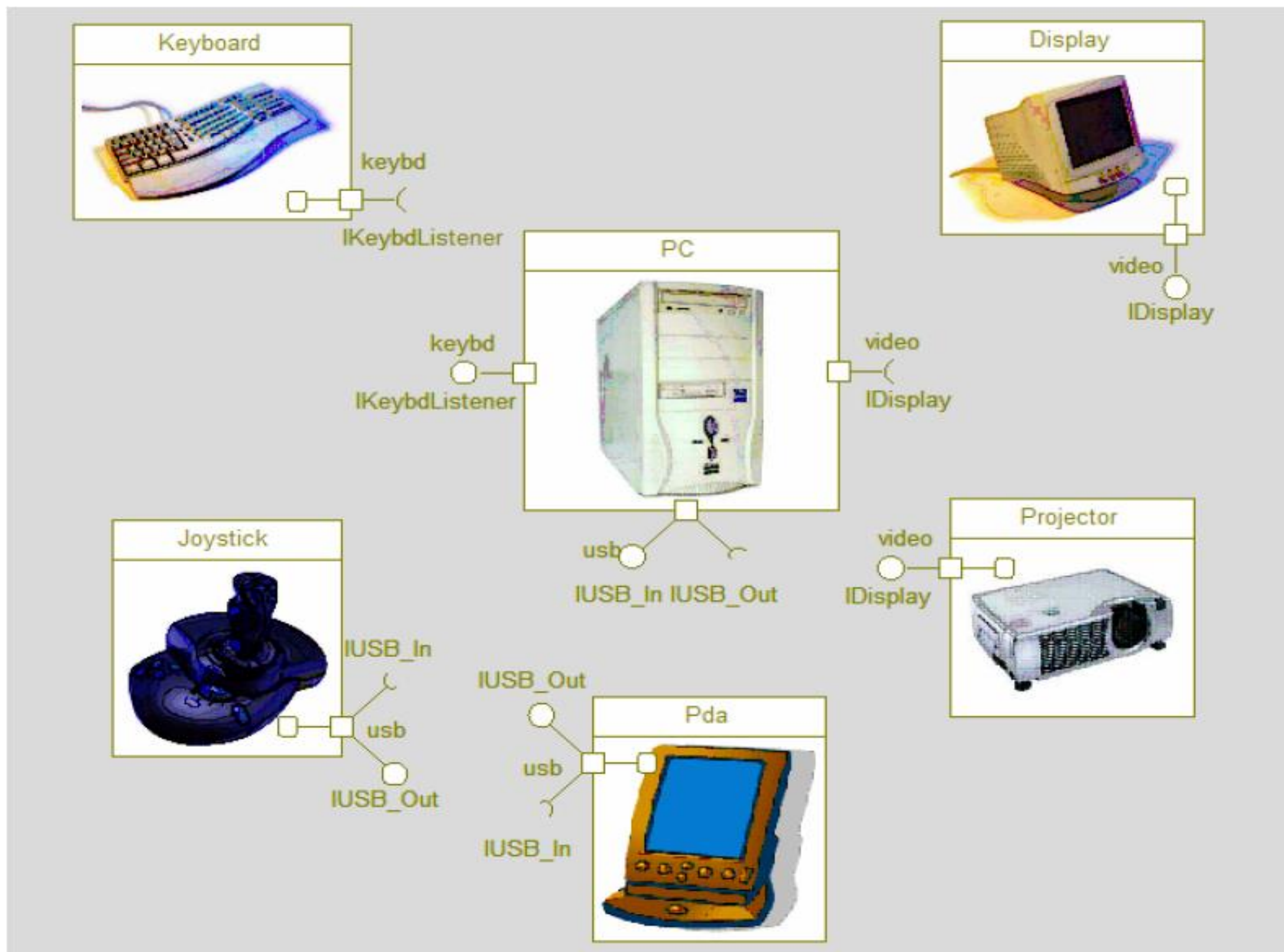


Provided/ Required Interface

- Provided interface
  - Class provides the services of the interface to outside callers
  - What the object can do
  - Provided interface accept incoming signal form outside callers
- Required interface
  - Class uses to implement its internal behavior
  - What the object needs to do
  - Outgoing signal are sent via required interface

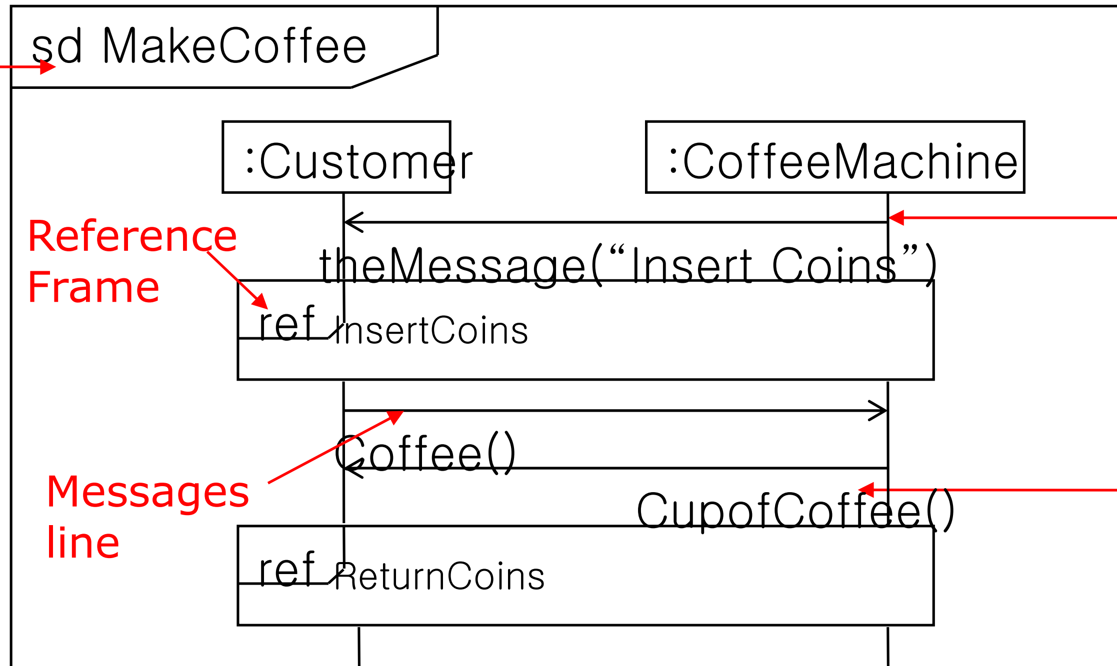


Computer Device Example



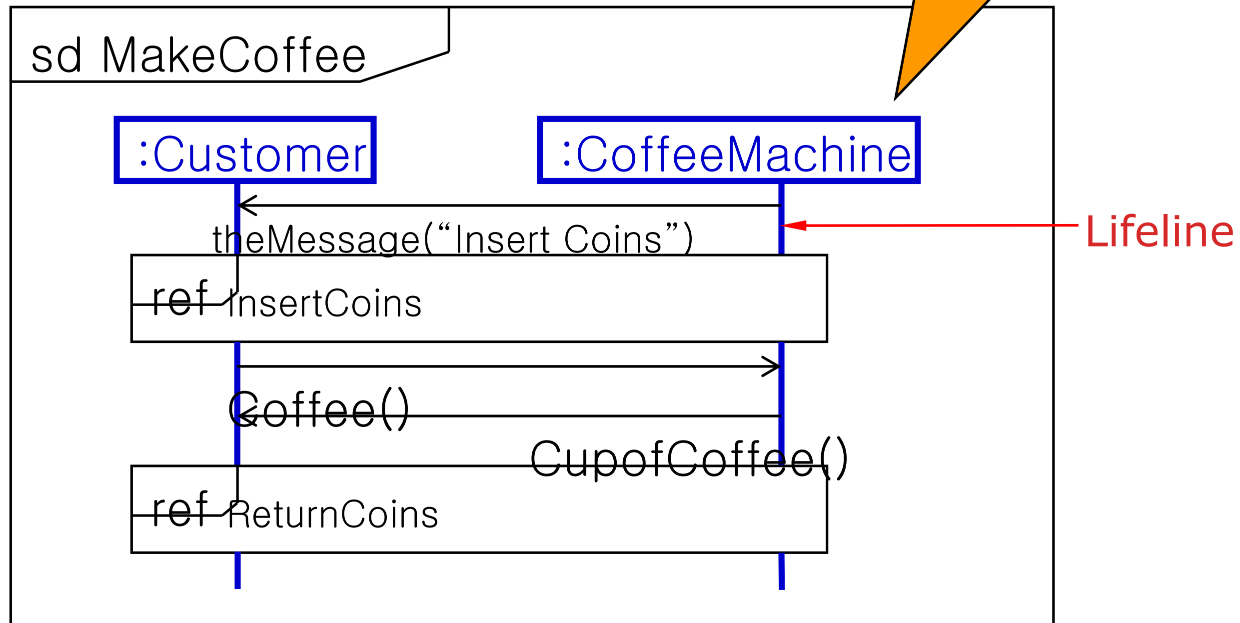
Sequence Diagram

- Emphasize on the sequence of communications between parts
- Show sequences of messages ("interactions") between instances in the system
- Emphasize time ordering



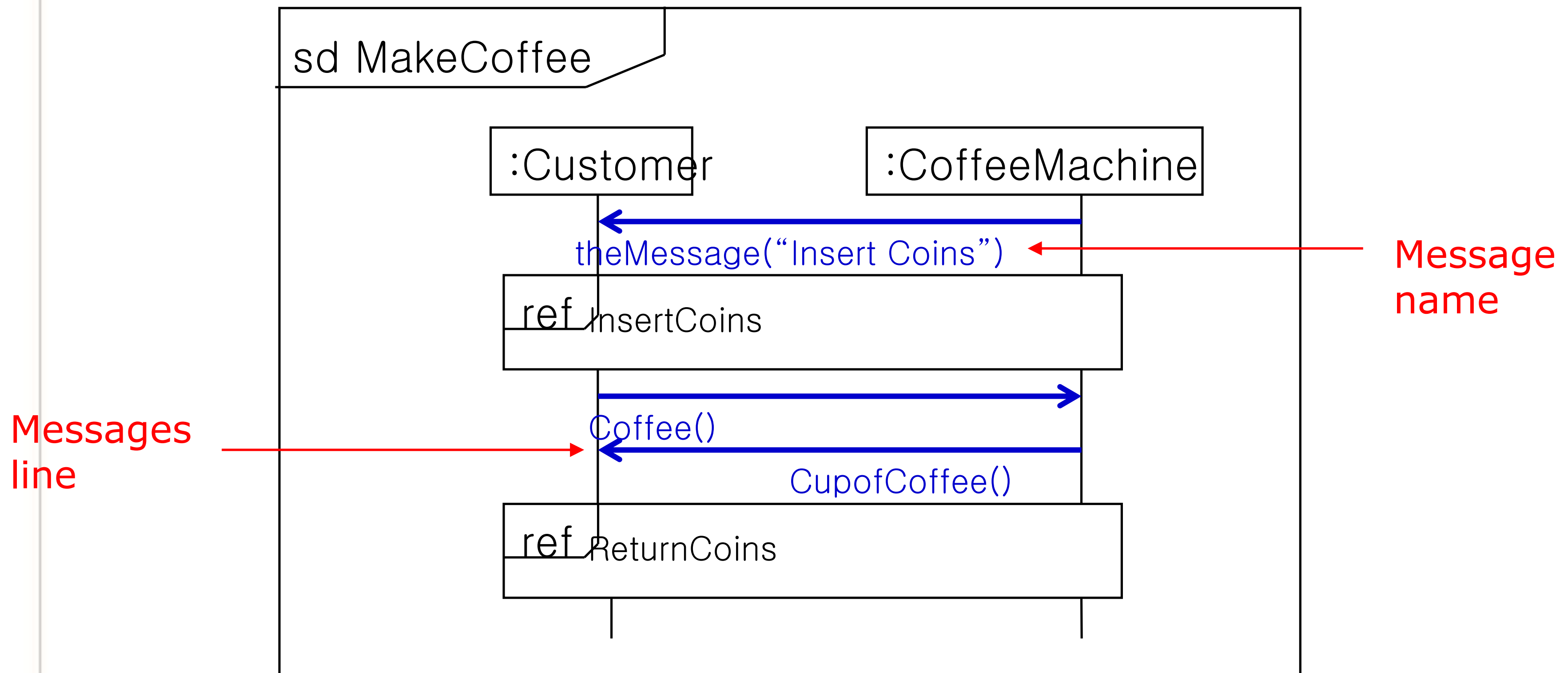
Lifelines

- Individual participant in the interaction over period time
  - Subsystem/ object/ class
  - Actor
  - External system roles in the interaction



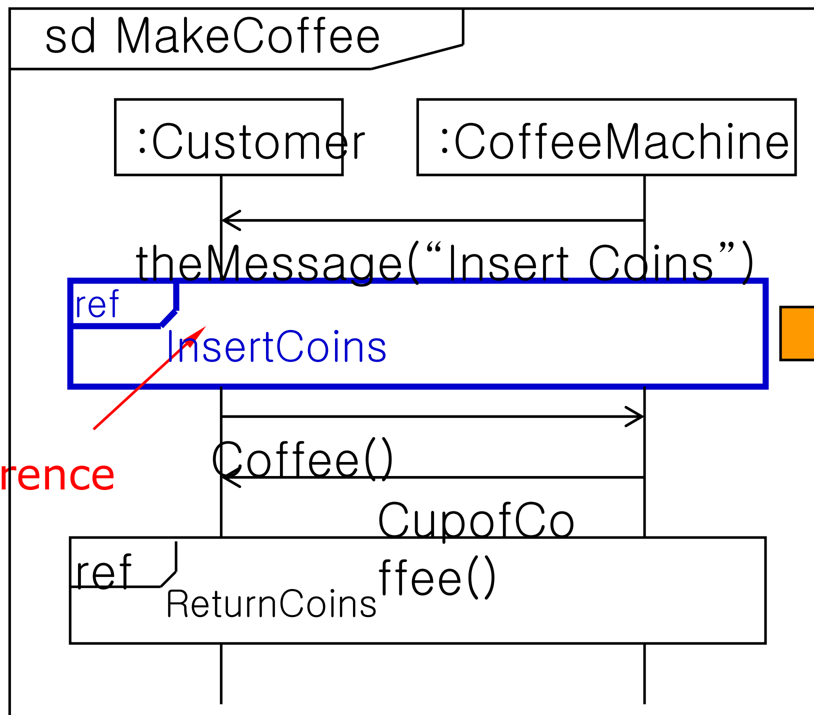
Messages

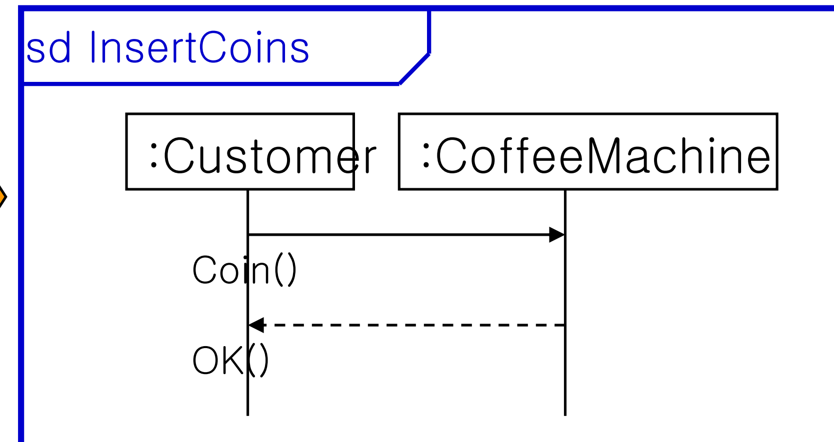
- One-way communication between two objects
- May have parameters that convey values



Referencing

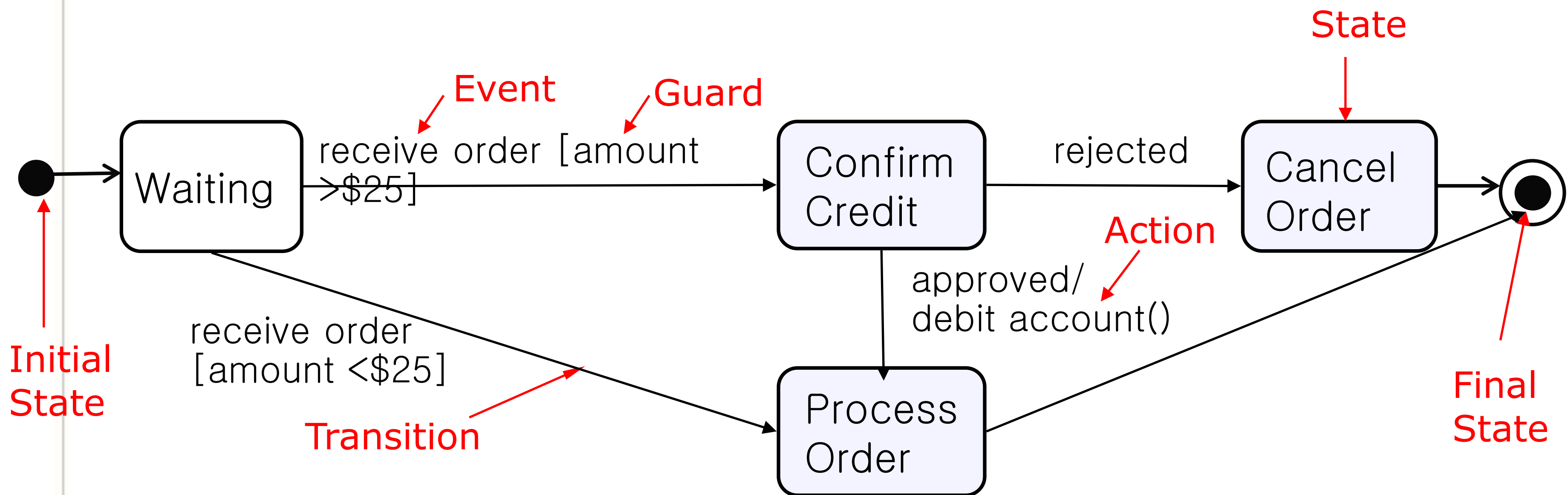
- Reuse already existing sequence diagrams
  - Avoid unnecessary duplication





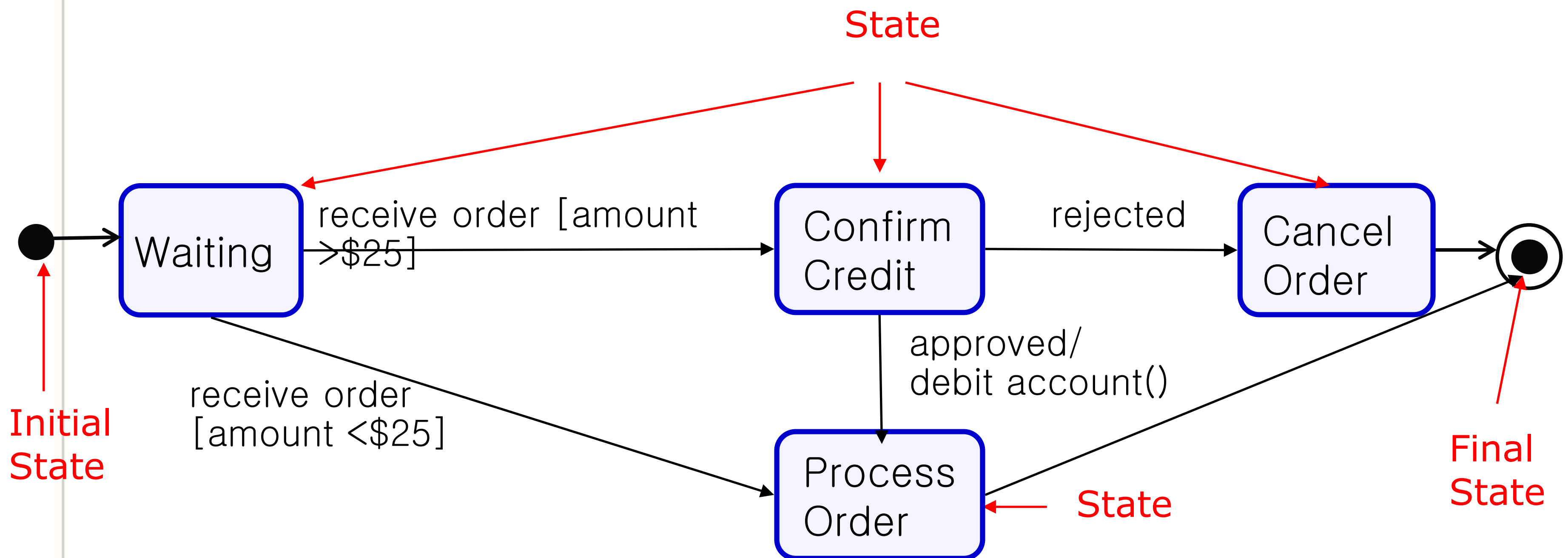
State Machine Diagram

- Specify the dynamic behavior of an element
- Show
  - The life history of a given class
    - Capture significant events that can act on an object
  - The event that cause a transition from one state to another
  - The actions that result from a state change



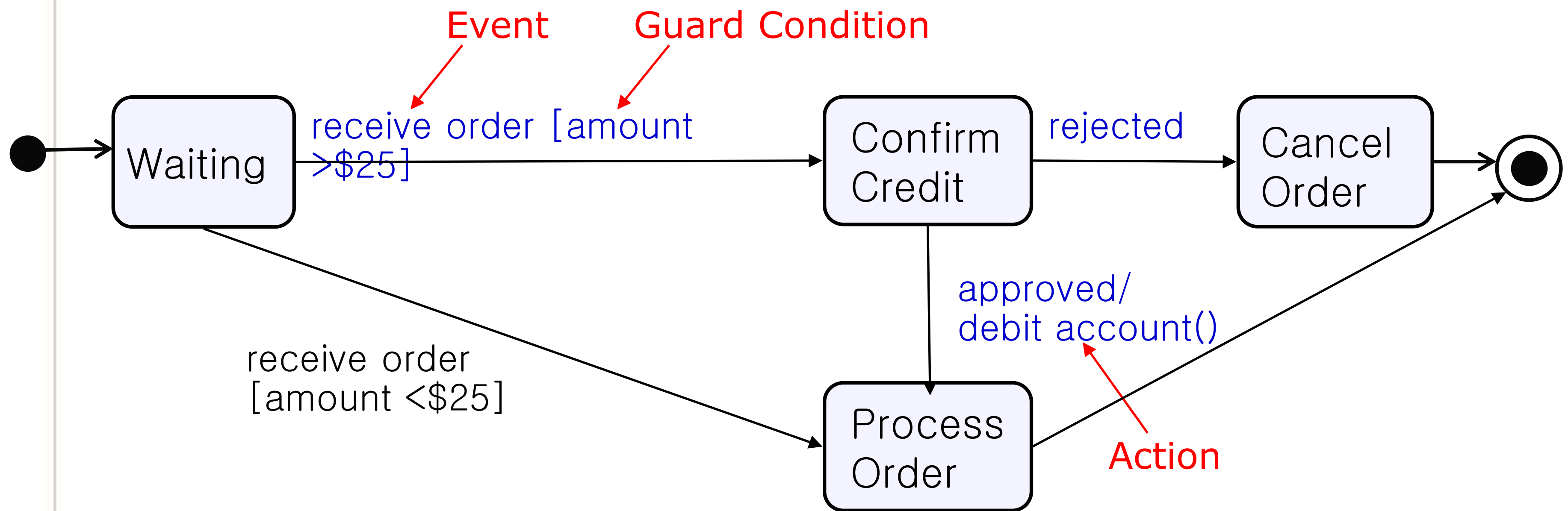
States

- State
  - Condition or situation during the life of an object
    - Satisfies some condition, performs some activity or waits for some event



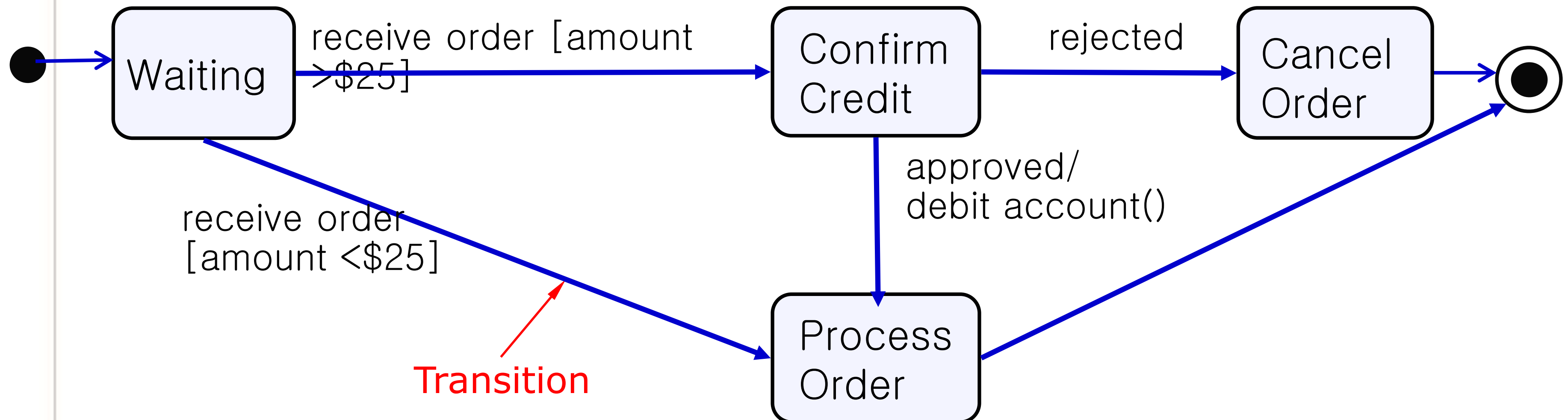
Event and Action

- Event
  - Stimulus which causes the object to change state
- Action
  - Output of a signal or an operation call

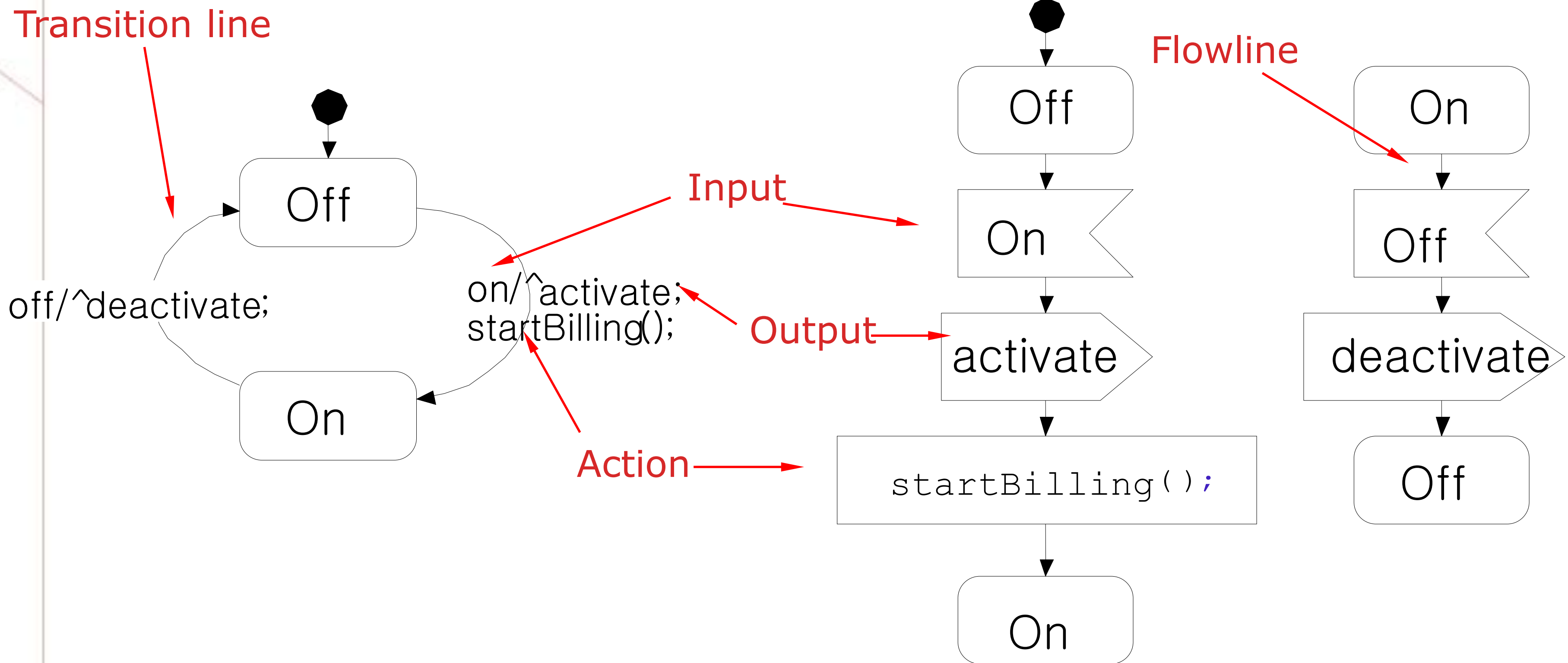


Transition

- Change state from one to another triggered by an event
- Occur only when guard condition is true
- Syntax: event(arguments)[condition]/action



State or Transition-oriented Syntax



- Transition line: transition details shown on line textually
- Flowline: simple line; transition details shown in chained symbols

Q & A