2018 Fall
CS350 Introduction to Software Engineering

Unified Modeling Language (UML)

Software Engineering Lab (SELab), KAIST

Outline

1. **Software System Modeling**

2. **Unified Modeling Language (UML)**
   A. Modeling Language (ML)
   B. UML Diagram Taxonomy
   C. Design Activities with UML

3. **UML Diagrams**
   A. Use Case Diagram: Modeling Requirements
   B. Class Diagram: Modeling System Structure
   C. Sequence Diagram: Modeling Ordered Interactions
   D. Other Diagrams

4. **UML Support Tools**

Software System Modeling

What is a Model?

- **A model is an intended simplification of reality.**

- **Models (i.e., specifications) describe structures and behaviors of a system they intend to model.**
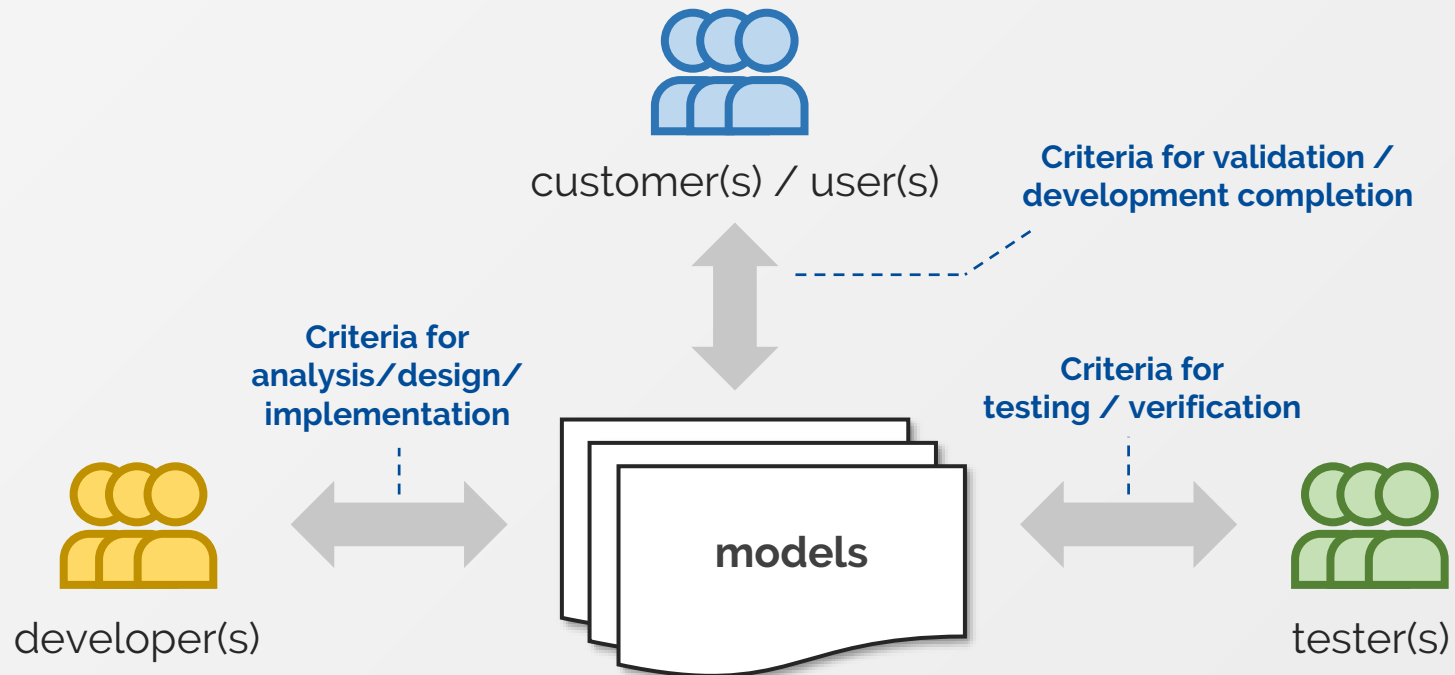

Modeling

Implementation

Modeling is a proven and well-accepted engineering technique
(e.g., Architectural model of houses and buildings, Mathematical models)

Why Do We Use Models for Software Development?

○ **To easily communicate information between different stakeholders in an unambiguous way**

- We can analyze and design a system in a more reliable and structured way by using software development.

customer(s) / user(s)

Criteria for validation / development completion

Criteria for analysis/design/ implementation

Criteria for testing / verification

models

developer(s)

tester(s)

Typical Properties of Good Models

Correct
Every statement is one that the software shall meet

Unambiguous
A model should have only one interpretation

Complete
Models should include all the requirements

Consistent
Subsets should not have any conflict

UML
Unified Modeling Language

Modeling Language (ML)

- **A modeling language is any language with which a model can be described.**
    - An ML can be graphical, textual, or more specific types
        - ▸ Graphical ML uses a diagram technique
        - ▸ Textual ML uses standardized keywords / expressions
    - An ML contains two elements:
        - ▸ **Notation** is the elements that make up a modeling language
        - ▸ **Semantics** are the descriptions of what the notation means

An ML can be anything that helps you describe your system.
pseudo-code, source code, pictures, diagrams, or descriptions
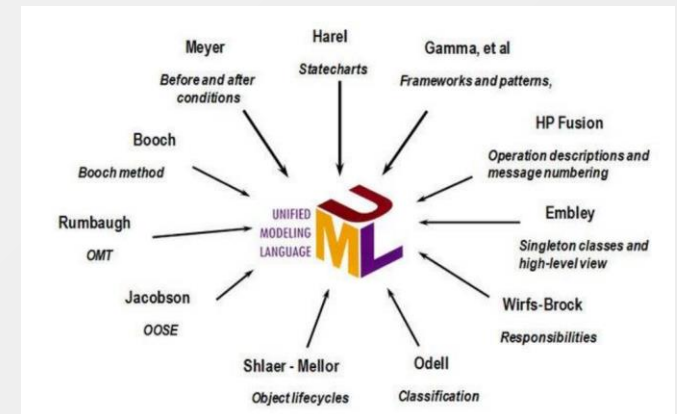
Unified Modeling Language (UML) (1/2)

- **Unified**
  - End to many similar approaches (i.e., modeling languages)
  - Standardized by Object Management Group (OMG)

- **Modeling**
  - Main creative process of software development

- **Language**
  - Standardized & graphical modeling languages to describe S/W
  - Collection of different diagram types
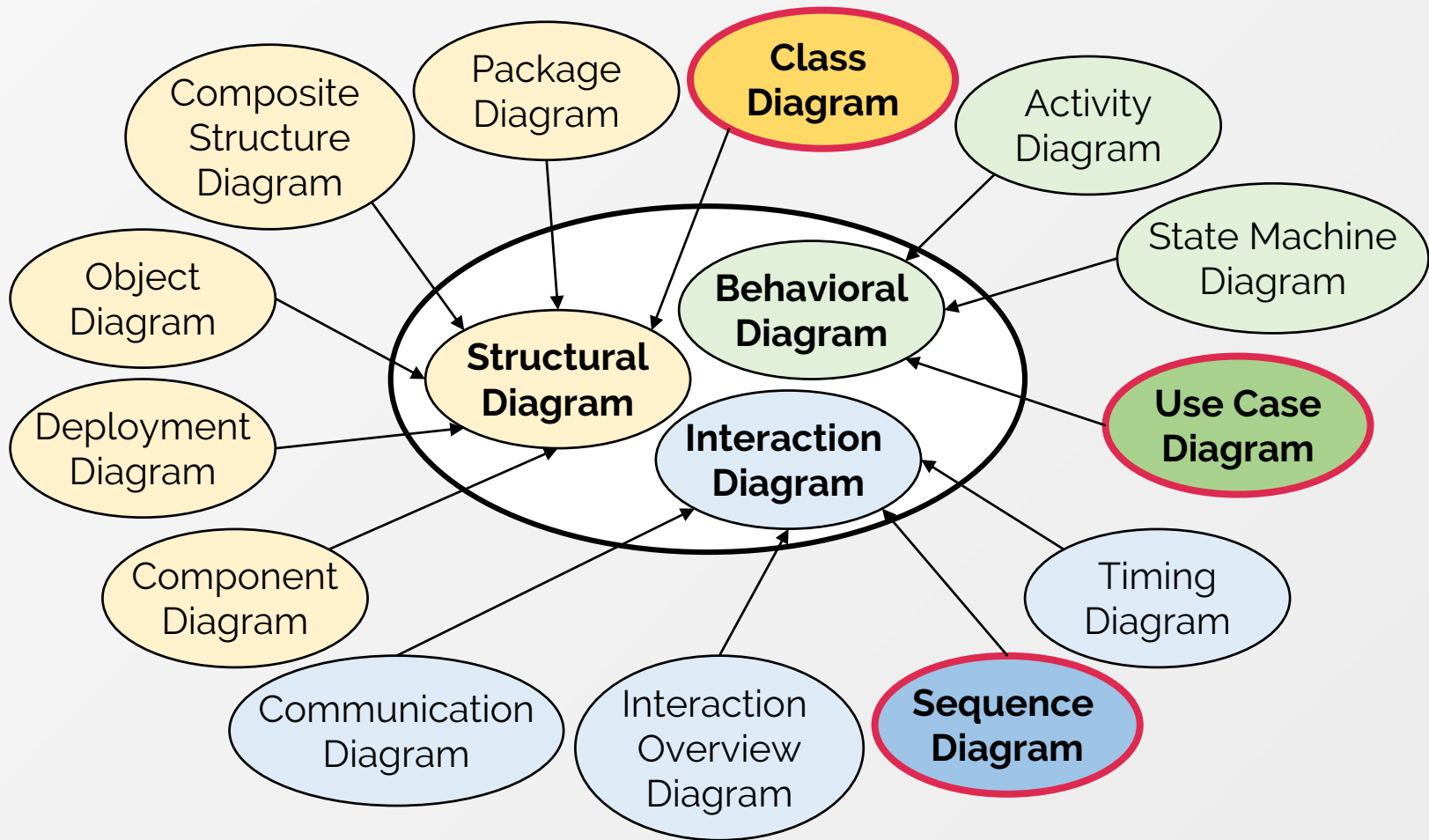
Unified Modeling Language (UML) (2/2)

○ **UML is a visual (graphical) modeling language for specifying, constructing and documenting:**

- Object-oriented modeling
- Model/view paradigm
- Target language (or technique) independent

○ **UML 2.0 leverages the industry's investment in UML 1.x and makes UML comprehensive, scalable and mature.**

- Latest version: UML 2.5 (June 2015)
    ▸ http://www.omg.org/spec/UML/

UML Diagrams for Software Modeling

○ **UML is a visual modeling language used to describe the model of software system.**

- **Structure**: What are software's structural features?
- **Behavior**: How do the software components interact?

Representation of structure

Representation of interactions

UML Diagram Taxonomy

UML Diagrams

Structural Diagrams of UML

Diagram	Description	Elements
Class diagram	Shows structure of the designed system, subsystem or component as related classes and interfaces, with their features, constraints and relationships	Class, interface, feature, constraint, relationships
Object diagram	Shows instance specifications of classes and interfaces (objects), slots with value specifications, and links (i.e., a class diagram with objects and no classes)	Instance specification, object, slot, link
Package diagram	Shows packages and relationships between the packages	Package, element, dependency, element/ package import & merge
Component diagram	Shows components and dependencies between them. - Used for Component-Based Development (CBD) - Used to describe systems with Service-Oriented Architecture (SOA)	Component, interface, provided & required interface, class, port, connector, artifact, etc.
Composite structure diagram	Show (a) internal structure of a classifier, (b) a behavior of a collaboration	
Deployment diagram	Show architecture of the system as deployment of software artifacts to deployment targets	Deployment, artifact, deployment target, node, device, execution, environment, communication path, deployment spec

Behavioral Diagrams of UML

Diagram	Description	Elements
Use-case diagram	Describe a set of actions (use cases) that some system or systems should or can perform in collaboration with one or more external users of the systems (actors) to provide some observable and valuable results to the actors or other stakeholders of the system	Use case, actor, subject, extend, include, association
State machine diagram	Used for modeling discrete behavior through finite state transitions and expressing the behavior of a part of the system	State, transition, pseudostate
Activity diagram	Shows sequence and conditions for coordinating lower-level behaviors, rather than which classifiers own those behaviors (e.g., control flow and object flow models)	Activity, partition, action, object, control, activity edge
Sequence diagram	Most common kind of interaction diagrams which focuses on the message interchange between lifelines (objects)	Lifeline, execution, specification, message, combined fragment, interaction use, state invariant, destruction occurence
Communication diagram, Interaction overview diagram, Timing diagram		*See Note.

In This Lecture,

○ **We will focus on two phases of software development:**

- Analysis phase, Design phase

○ **We will study three types of UML diagrams:**

- Use-case diagram
  - ▸ A high level visualization of how the system works
  - ▸ Built on requirement specifications from discussions with developers, customers, and/or end users
- Class diagram
  - ▸ A collection of static model elements such as classes and types, their contents, and their relationships
- Sequence diagram
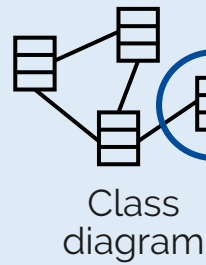 - ▸ Model of sequential logic, in effect the time ordering of messages between classifiers

In This Lecture,

UML Diagrams (1/3)

Use Case Diagram
Modeling Requirements

SELAB
KAIST

Software Requirements

- **Requirements are descriptions of the services that a software system must provide and the constraints under which it must operate[1].**
  - Requirements describe what the system will do at a high-level.
  - Requirements Engineering (RE) is the process of establishing the needs of stakeholders and the constraints.
    - ▸ Requirements elicitation, definition, and so on.

Requirement A.1
The content management system shall allow an administrator to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database.

Example functional requirements of Weblog content management system

Actor & Use Case

Actor

- Someone or something that must interact with the system under development
    - ▸ Users, external systems, devices, etc.
- Not part of the system under development

Use Case

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

Steps for Requirements Modeling

I. **Capture system requirements**
- Elicit requirements from stakeholders
- Analyze requirements

II. **Describe system requirements**
- Capture actors & use cases
- Connect communication lines
- Draw system boundaries

III. **Describe use cases**
- Complete use case descriptions

Step 1: Capturing a System Requirement

- **Elicit requirements from stakeholders and analyze the requirements**
  - Example: Weblog Content Management System (Weblog CMS)

Requirement A.1
The content management system shall allow an administrator to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database.

There is no specific best way to start analyzing requirements, but we can look at **things that interact with the system** for starters.
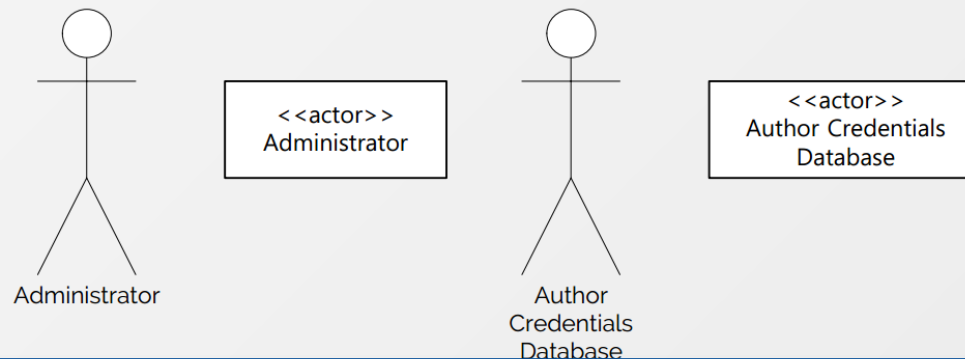
Step 2: Describing System Requirements (1/4)

○ Capture actors from requirements

- Someone or something interacting with the system, but not part of the system under development

Requirement A.1

The content management system shall allow an administrator to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database.

<<actor>>
Administrator

<<actor>>
Author Credentials
Database

Administrator

Author
Credentials
Database

How to identify an actor? **See appendix**

Step 2: Describing System Requirements (2/4)

- **Capture use cases from requirements**
  - Use cases where the system is being used to complete a specific job for an actor

Requirement A.1

The content management system shall allow an administrator to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database.

Create a
New Blog Account

usually starts with a **verb**
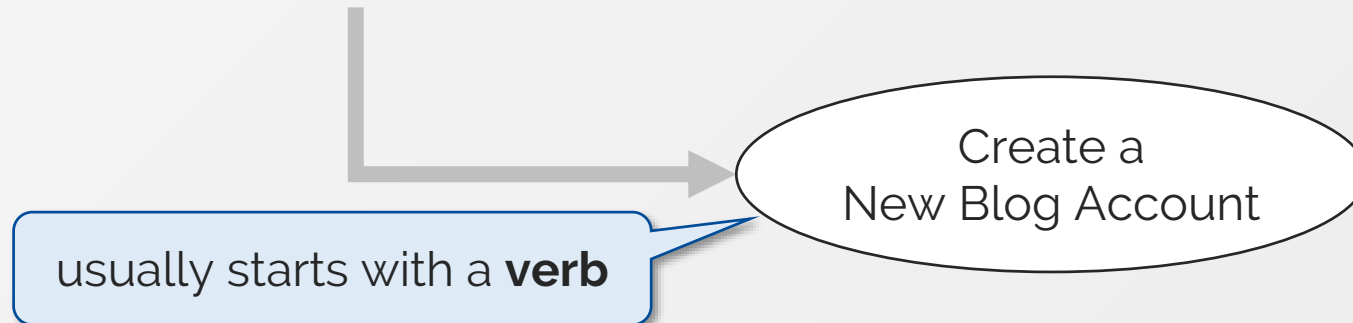
Step 2: Describing System Requirements (3/4)

○ Connect **communication lines**

- Connection between an actor and a use case to show the actor participating in the use case

| **Requirement A.1** |
|---|
| The content management system shall allow an administrator to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database. |

Administrator   Author credentials DB

A communication line between an actor (Administrator) and a use case

Step 2: Describing System Requirements (4/4)

◉ Draw **system boundaries**

- Explicit system boundary with the name of your system (or subsystem) for separating actors and use cases
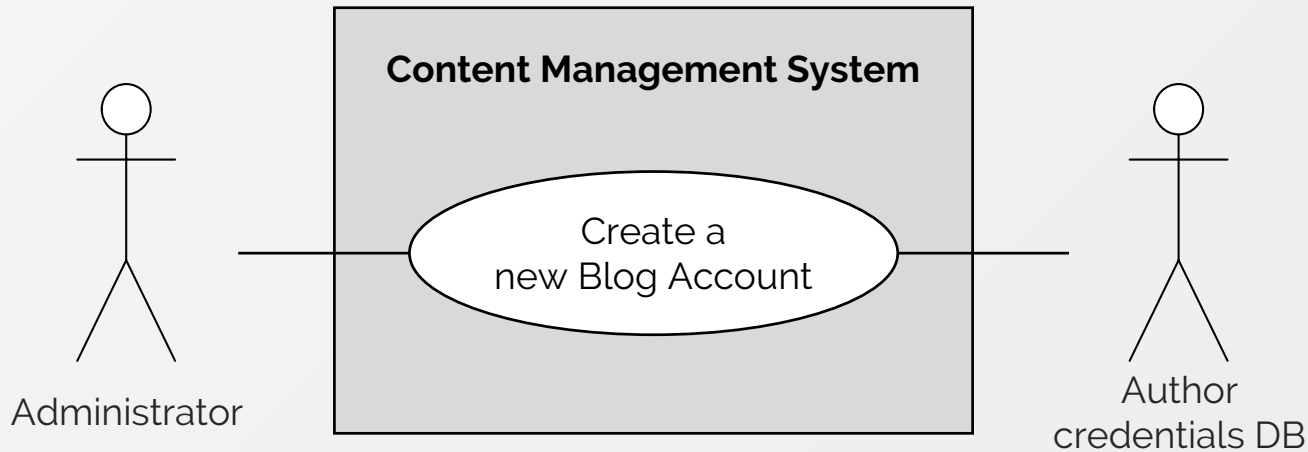
Step 3: Describing Use Cases

⊙ **Complete use case descriptions to express important information in the form of a text-based description**
- They provide enough detail to system designers

| Use case description detail	What the detail means and why it is useful
Related Requirements	Some indication as to which requirements this use case partially or completely fulfills.
Goal In Context	The use case's place within the system and why this use case is important.
Preconditions	What needs to happen before the use case can be executed.
Successful End Condition	What the system's condition should be if the use case executes successfully.
Failed End Condition	What the system's condition should be if the use case fails to execute successfully.
Primary Actors	The main actors that participate in the use case. Often includes the actors that trigger or directly receive information from a use case's execution.
Secondary Actors	Actors that participate but are not the main players in a use case's execution.
Trigger	The event triggered by an actor that causes the use case to execute.
Main Flow	The place to describe each of the important steps in a use case's normal execution.
Extensions	A description of any alternative steps from the ones described in the Main Flow.

SELAB
KAIST

Example: Weblog CMS

Requirements & Use case diagram

Requirement A.1
The content management system shall allow an administrator to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database.

Requirement A.2
The content management system shall allow an administrator to create a new personal Wiki, provided the personal details of the applying author are verified using the author credentials database.
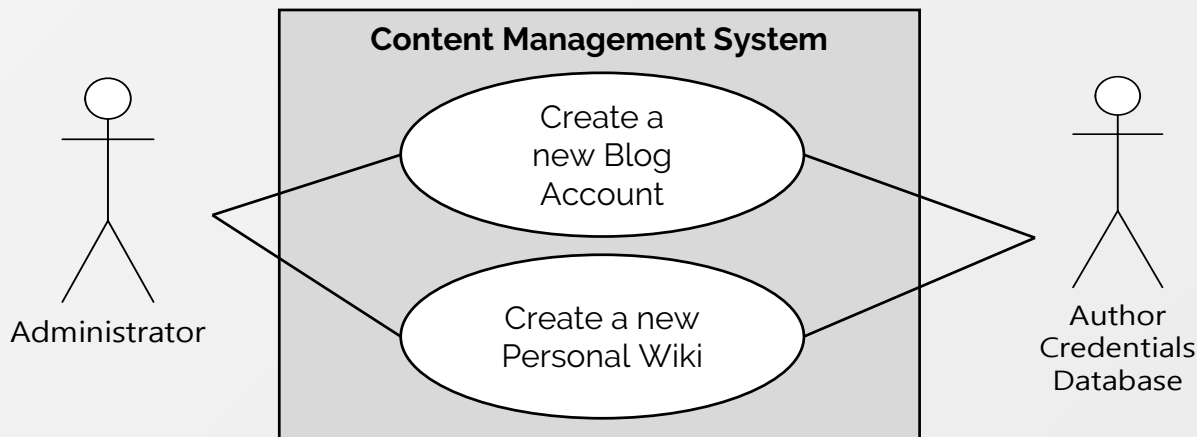
Example: Weblog CMS

○ Use case description of <Requirement A.1> (1/2)

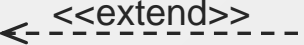| Use case name	Create a new Blog Account
Related Requirements	Requirement A.1.
Goal In Context	A new or existing author requests a new blog account from the Administrator.
Preconditions	The system is limited to recognized authors and so the author needs to have appropriate proof of identity.
Successful End Condition	A new blog account is created for the author.
Failed End Condition	The application for a new blog account is rejected.
Primary Actors	Administrator.
Secondary Actors	Author Credentials Database.
Trigger	The Administrator asks the CMS to create a new blog account.
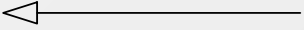
Example: Weblog CMS
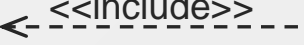
○ Use case description of <Requirement A.1> (2/2)

Use case name	Create a new Blog Account	
Main Flow	**Step**	**Action**
	1	The Administrator asks the system to create a new blog account.
	2	The Administrator selects an account type.
	3	The Administrator enters the author's details.
	4	The author's details are verified using the Author Credentials Database.
	5	The new blog account is created.
	6	A summary of the new blog account's details are emailed to the author.
Extensions	**Step**	**Branching Action**
	4.1	The Author Credentials Database does not verify the author's details.
	4.2	The author's new blog account applications is rejected.

Use Case Relationships

- **Use case relationships provide your system designers with some architectural guidance**
  - So they can efficiently break down the system's concerns

- **UML provides 5 relationship types in a use case diagram**
  - Association between actor and use case
  - Generalization of an actor
  - Extend between two use cases
  - Include between two use cases
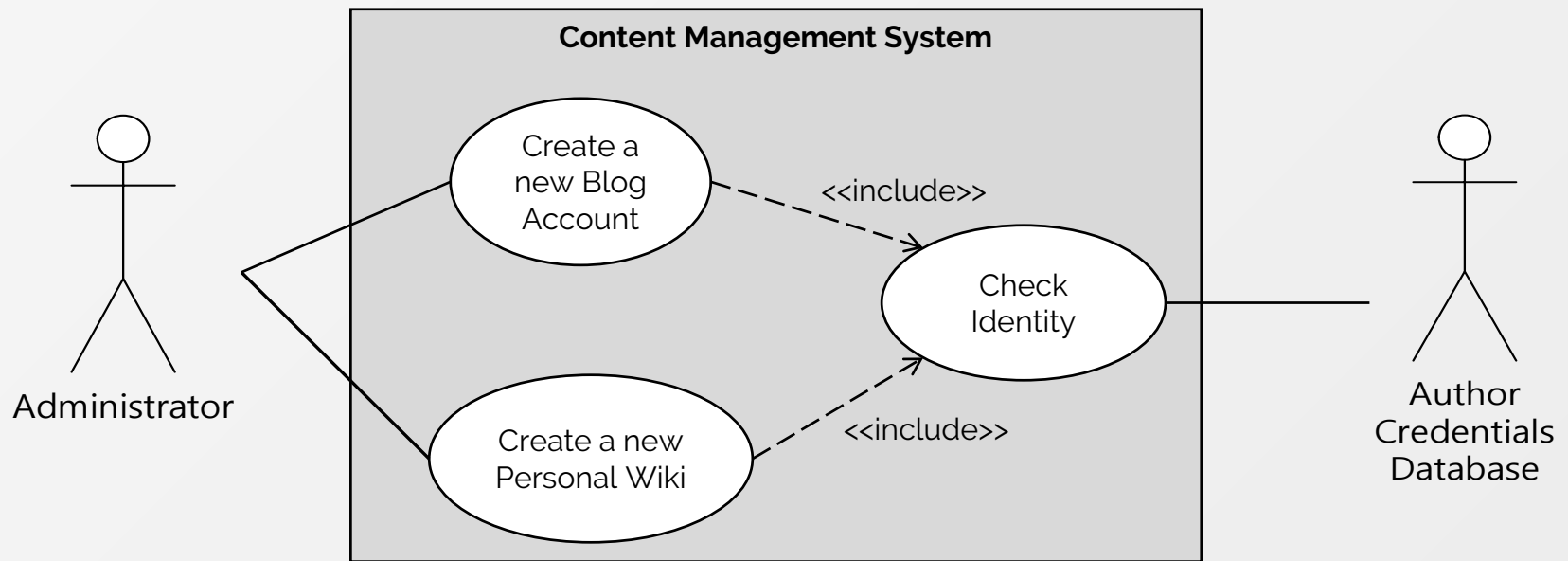 - Generalization of a use case

Use Case Relationships: <<include>>

○ **A use case can be reused by multiple use cases using the <<include>> relationship.**
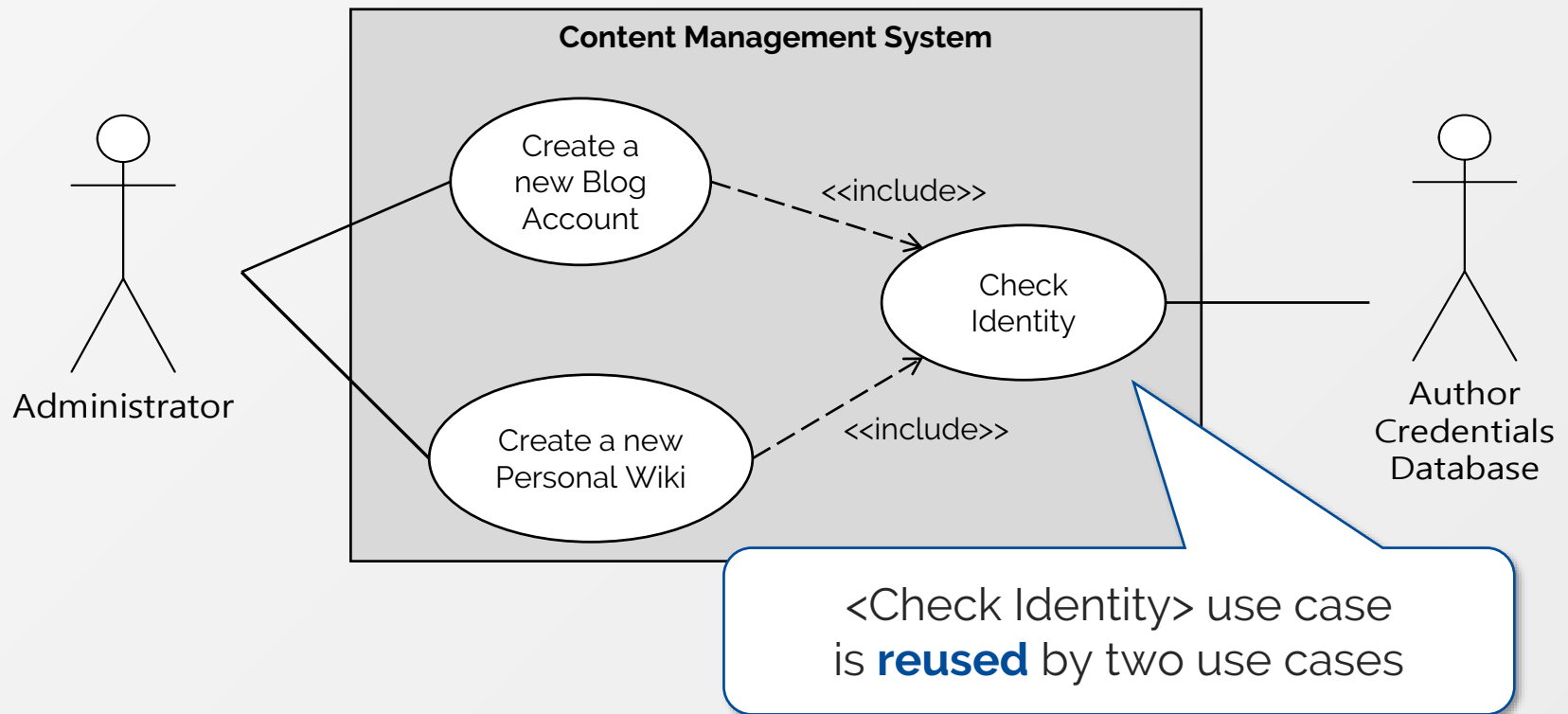
Use Case Relationships: <<include>>

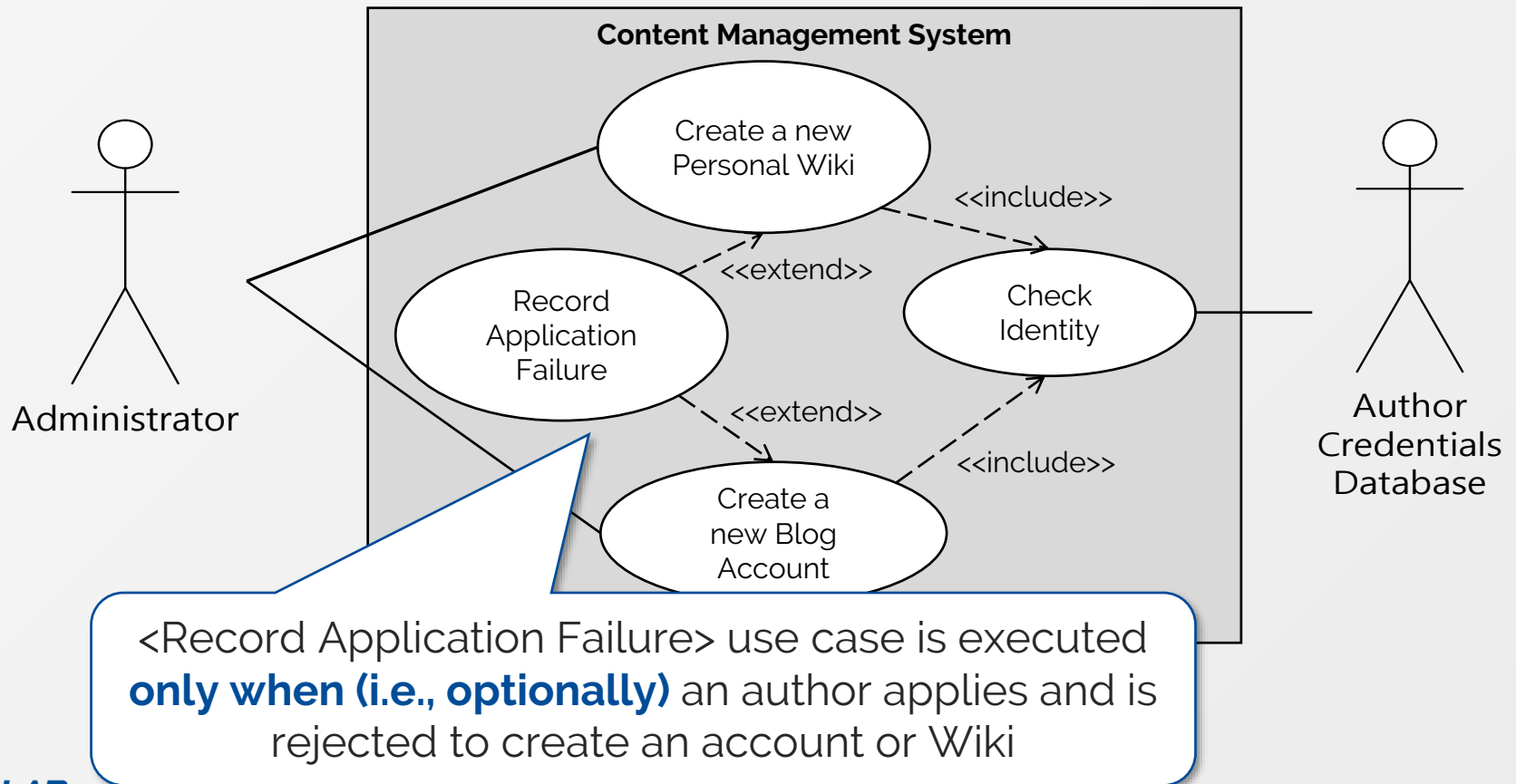- **A use case can be reused by multiple use cases using the <<include>> relationship.**

<use case A> - - - <<include>> - - -> **<use case B>**

<use case A> completely reuses all of the steps from the <use case B> being included

Use Case Relationships: <<extend>>

—

⊙ **The <<extend>> relationship specifies optional reuse depending on a runtime or system implementation decision.**

Content Management System

Administrator

Create a new Personal Wiki

<<include>>

<<extend>>

Record Application Failure

Check Identity

<<extend>>

Create a new Blog Account

<<include>>

Author Credentials Database

<Record Application Failure> use case is executed **only when (i.e., optionally)** an author applies and is rejected to create an account or Wiki
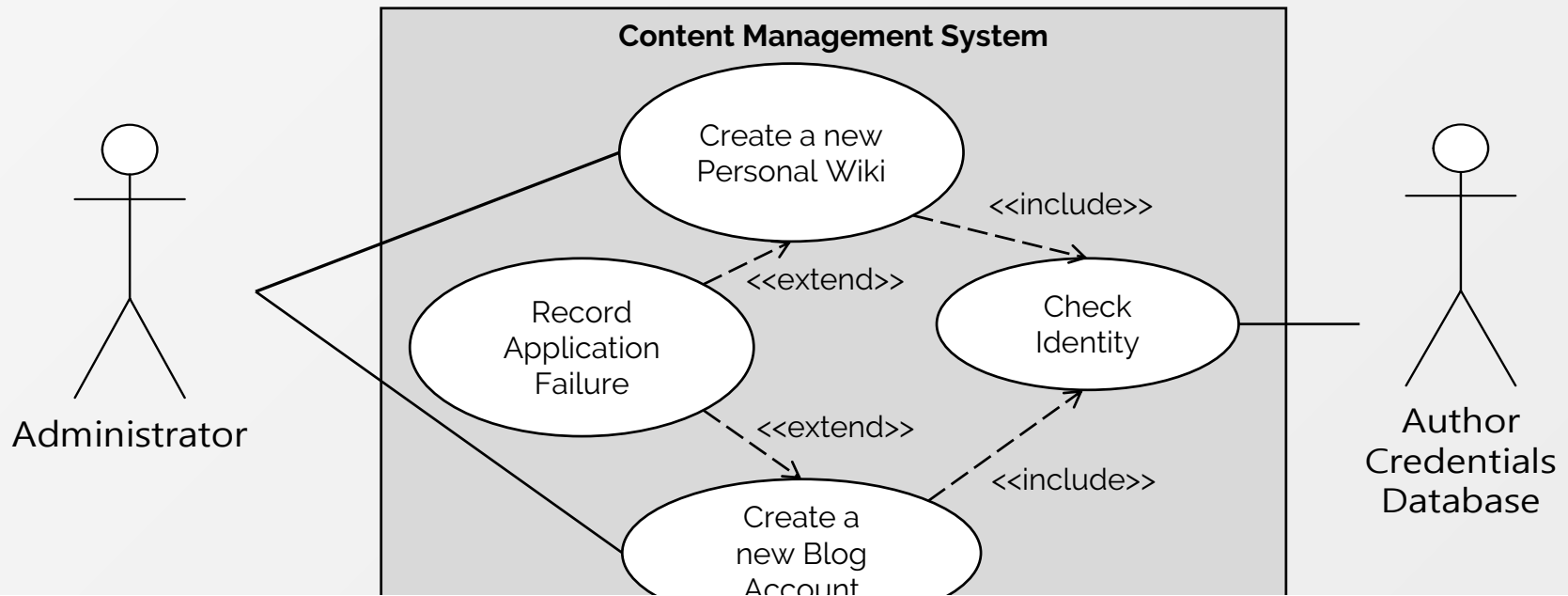
SE **LAB** KAIST

Use Case Relationships: <<extend>>

- **The <<extend>> relationship specifies optional reuse depending on a runtime or system implementation decision.**

<use case A> - - - <<extend>> - - -> **<use case B>**

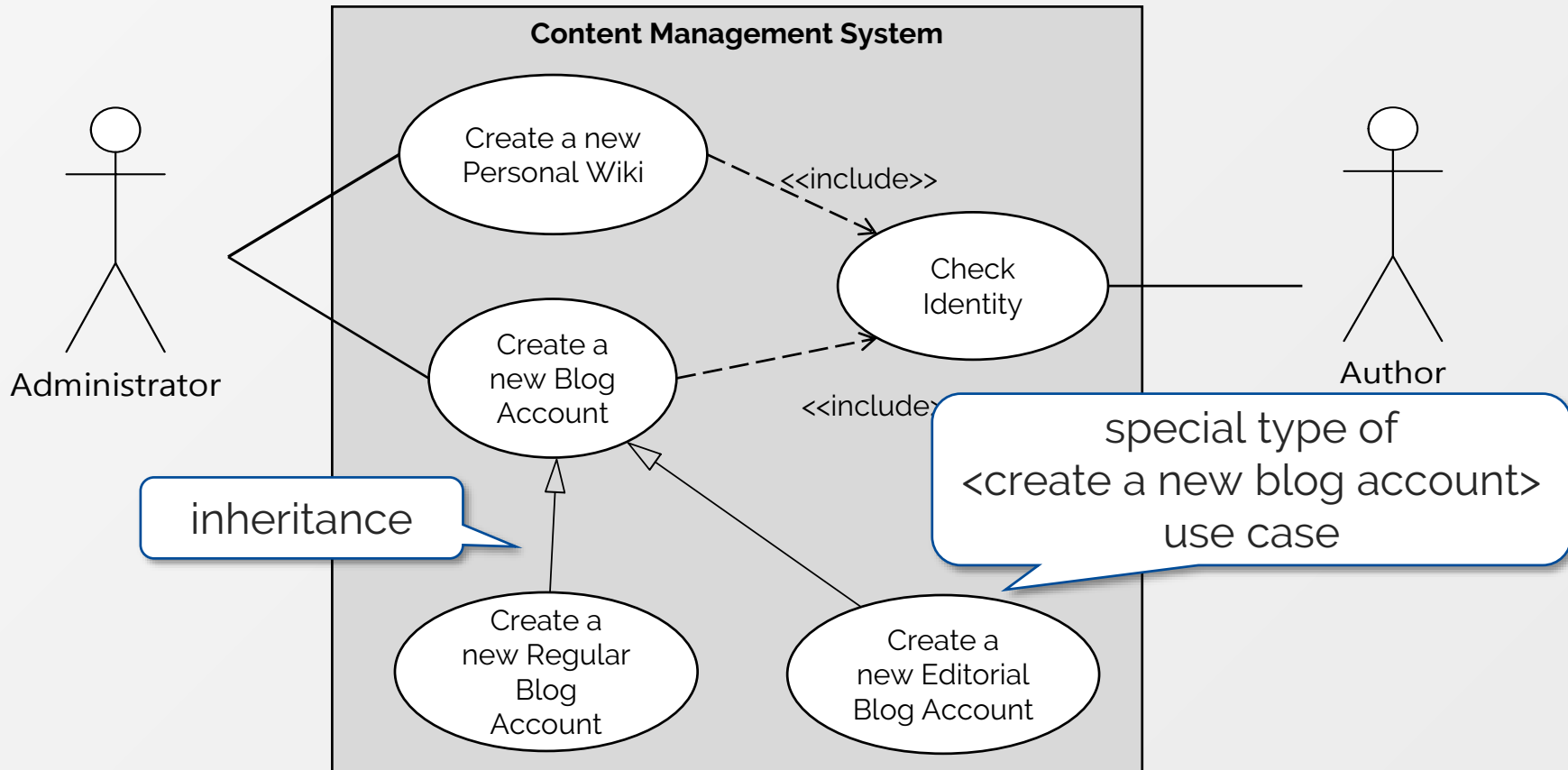<use case A> extends the behavioral options of <use case B> being extended

Use Case Relationships: Generalization
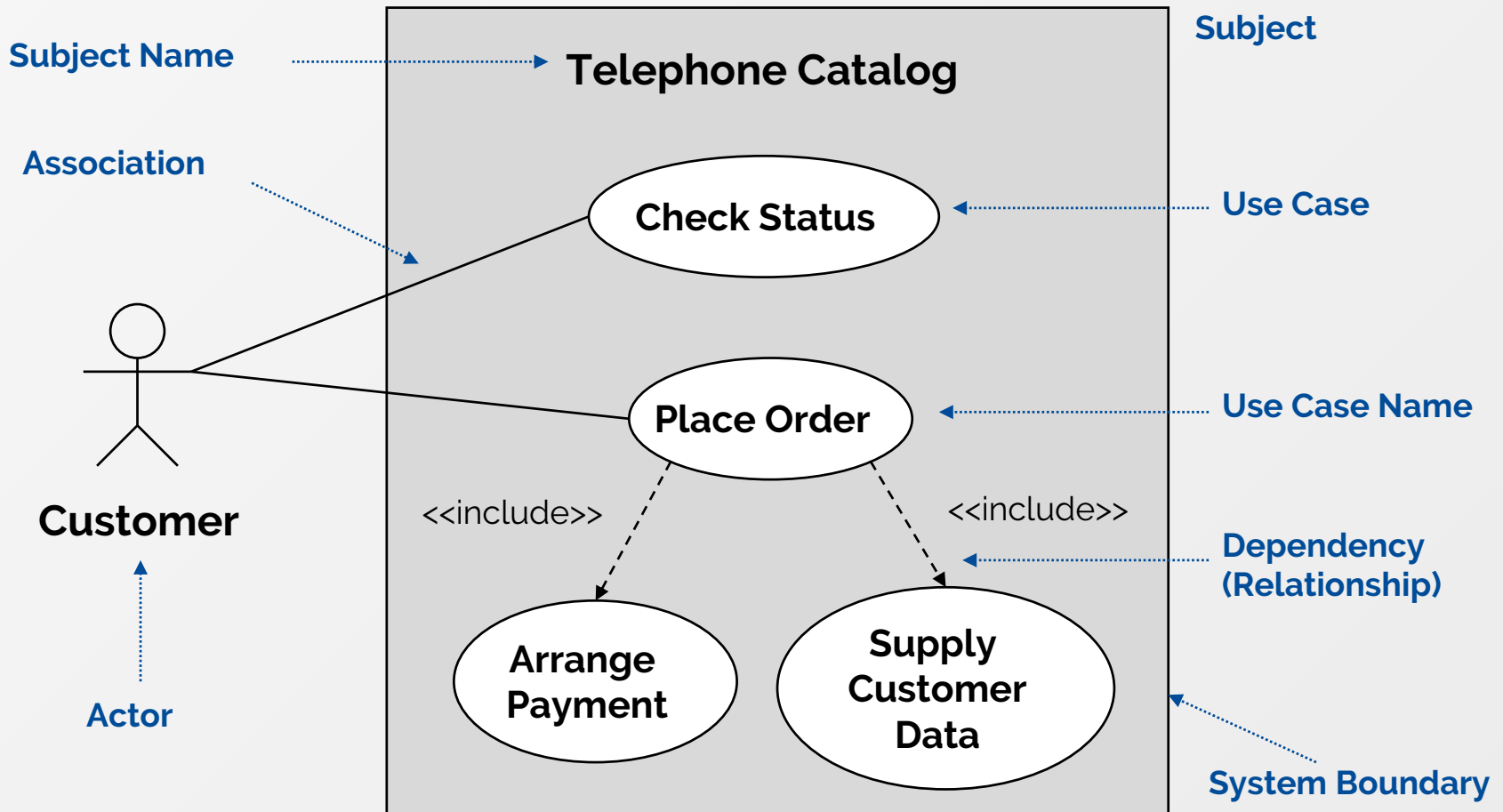
- **Generalization is useful when you want to show that one use case is a special type of another use case.**

Use Case Diagram Summary

Example use case

UML Diagrams (2/3)

Class Diagram
Modeling System Structure

Class of Object-Oriented Programming

- **Classes are basic building blocks of any OO system.**
  - A system structure is made up of a collection of pieces often referred to as objects.
  - An instance of a class is an object.

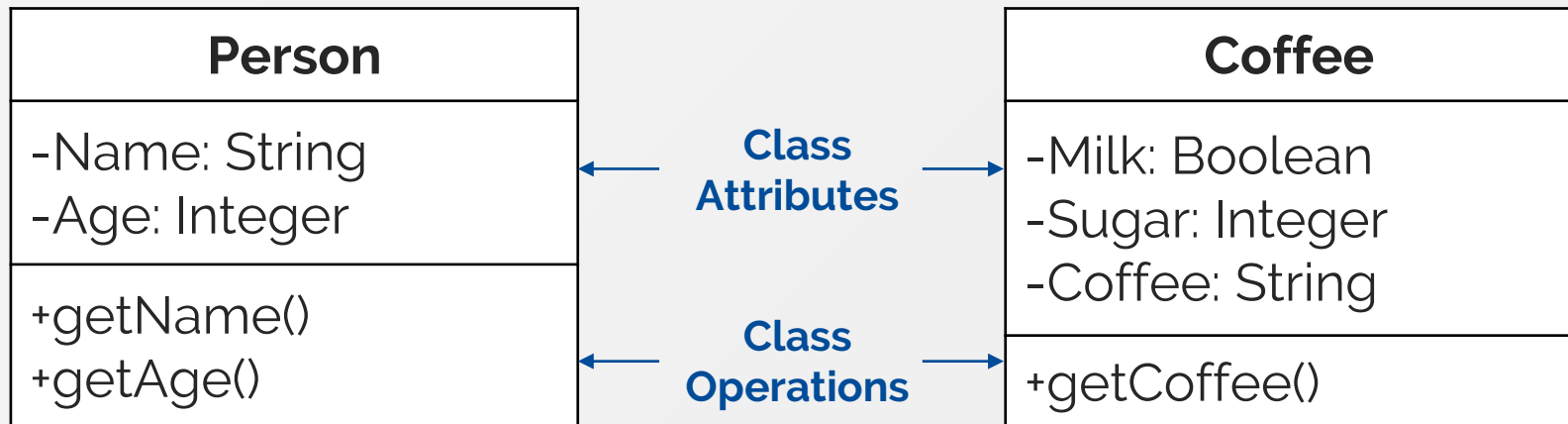| Class	Instantiate →	Object

An **instance** of a class

- **By defining classes, we can specify the types of objects in a system and the relationships between them.**
  - Abstraction, Encapsulation

State and Behavior of a Class

⊙ **A class describes attributes and operations to represent state and behavior, respectively**

- They enable a class to describe a group of parts within your system that share common characteristics

Person
-Name: String -Age: Integer
+getName() +getAge()

← **Class Attributes** →

← **Class Operations** →

Coffee
-Milk: Boolean -Sugar: Integer -Coffee: String
+getCoffee()

Steps for Class Modeling

I. **Find possible system classes**
   - Analyze classes of your system
   - Draw class boxes

II. **Describe class states & behaviors with attributes & operations**

III. **Specify visibility to enforce encapsulation**
   - Analyze the accessibility / visibility of attributes & operations
   - Describe visibility with symbols

IV. **Describe relationships between classes**

V. **Describe additional characteristics, dependencies**

Step 1: Find System Classes

⊙ **You can split into up to three sections to describe a class**

- Top section: name of the class
- Middle section: attributes of the class (optional)
- Bottom section: operations of the class (optional)

ClassName
Attribute Attribute
Operation Operation

ClassName
Attribute Attribute

ClassName
Operation Operation

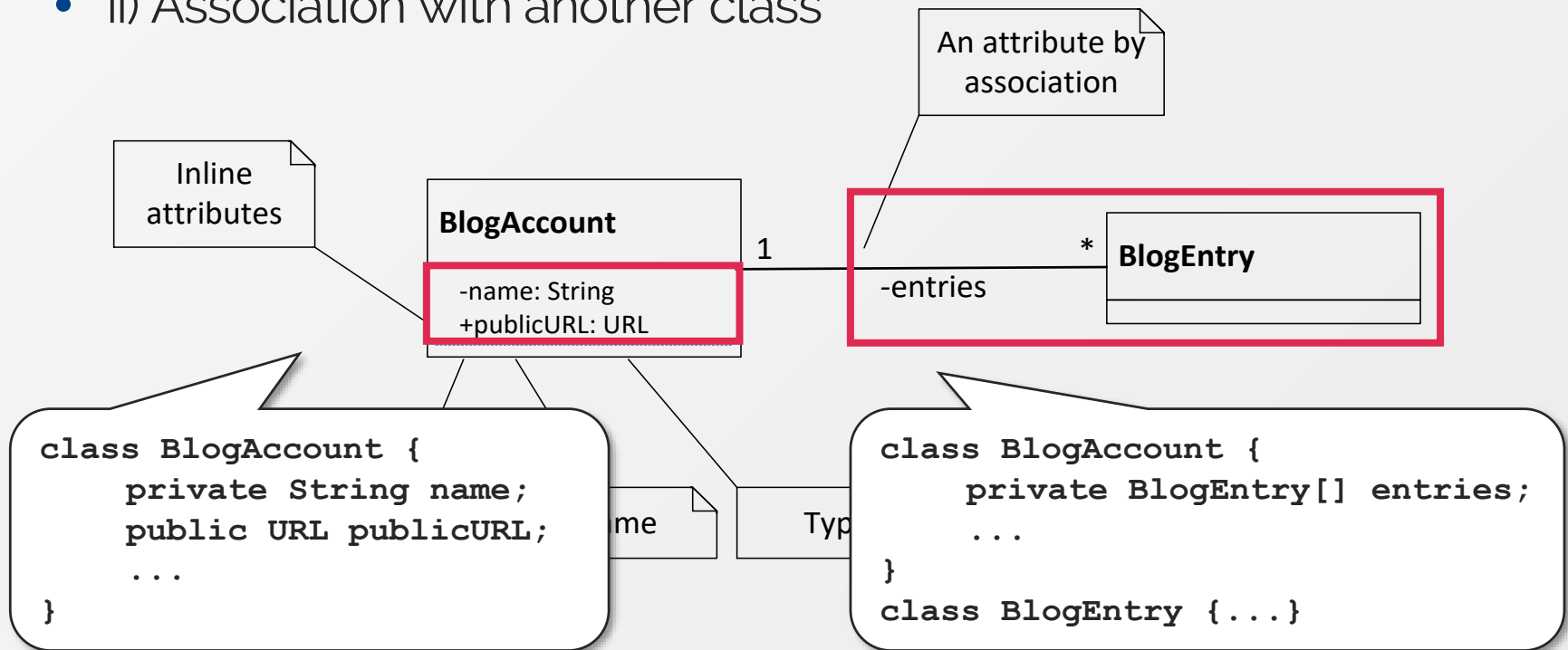| **ClassName** |
| --- |
| |
| |

**If some sections are not shown,
it does not imply that they are empty.**

Step 2: Specify Attributes & Operations (1/2)

○ **Attributes can be represented in two ways:**

- i) Placing them inside their section of the box (inline attributes)
- ii) Association with another class

An attribute by association

Inline attributes

BlogAccount

-name: String
+publicURL: URL

1

* **BlogEntry**

-entries

```
class BlogAccount {
    private String name;
    public URL publicURL;
    ...
}
```

```
class BlogAccount {
    private BlogEntry[] entries;
    ...
}
class BlogEntry {...}
```

Step 2: Specify Attributes & Operations (2/2)

⊙ **Operations are specified on a class diagram with a signature that is at minimum made up of:**

- Visible property, Name, Pair of parentheses for parameters, Return type
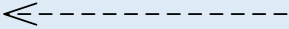   - ▷ +addEntry (newEntry: BlogEntry, author: Author): Void

```
┌─────────────────────────────────────────────────┐
│                  BlogAccount                      │
├─────────────────────────────────────────────────┤
│ - name: String                                    │
│ + publicURL: URL                                  │
│ - authors: Author [1..5]                          │
│ + addEntry(newEntry: BlogEntry, author: Author): Void │
└─────────────────────────────────────────────────┘
```

An operation

Visibility   Name   Parameters   Return Type
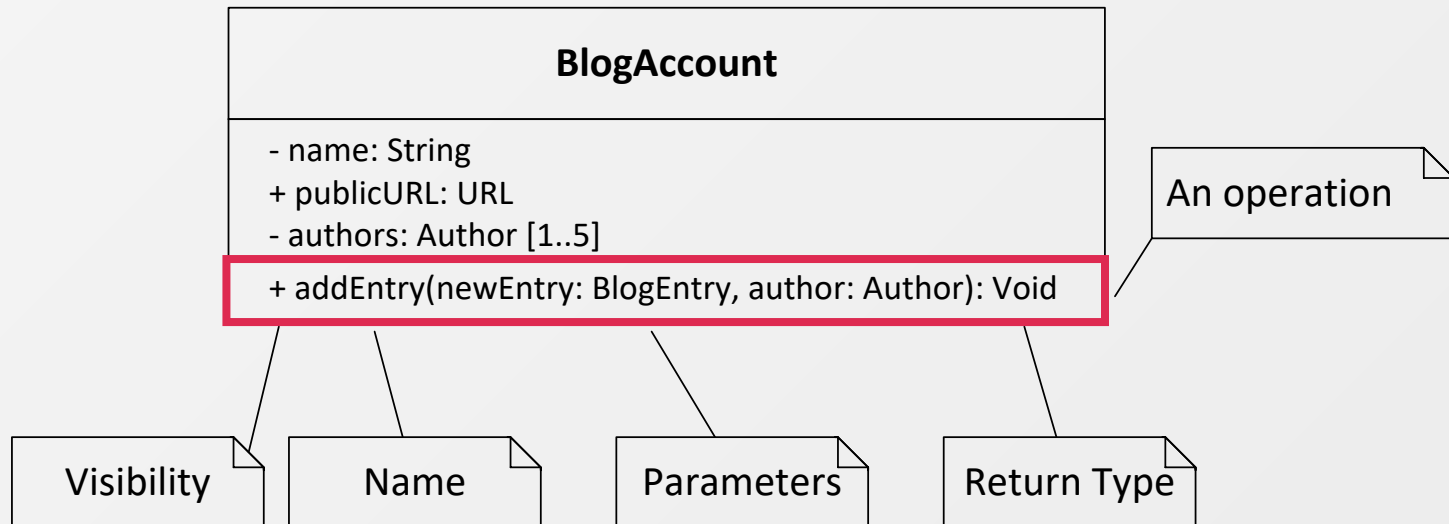
Step 3: Describe Visibility

—

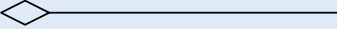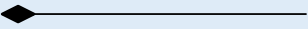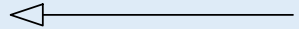○ **To enforce *encapsulation*, we use visibility to make a class selectively reveal its operations and data to other classes.**

| Visibility	Notation	Accessibility
Public	+	It can be accessible directly by any other class
Protected	#	It can be accessible by specialized classes (Part of the same class OR any other classes that inherit from the class)
Package	~	It can be accessible by any class in the same package
Private	–	It can be accessible within the declared class itself

Step 4: Describe Relationships between Classes

○ **We can describe relationships between classes using relationship arrows offered by UML.**

Weaker Class Relationship ⟵

⟶ **Stronger** Class Relationship

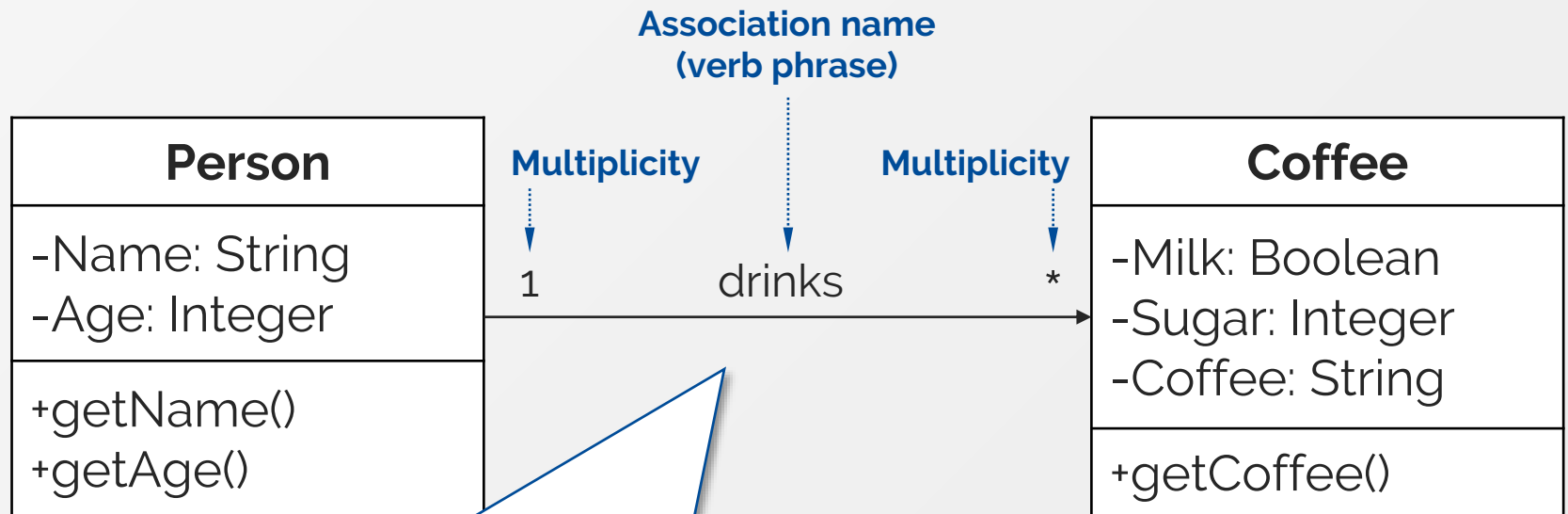| Dependency	Association	Aggregation	Composition	Inheritance
Dashed Arrow	Simple Connecting Line	Empty Diamond Arrow	Filled Diamond Arrow	Empty Arrow
When objects of one class **work briefly with** objects of another class.	When objects of one class **work with** objects of another class for some prolonged amount of time.	Weak 'whole-part" relationship **('has a')**	Strong "whole-part" relationships **('contains a')**	Relationships between superclass & subclasses
		When one class **owns but shares a reference** to objects of another class.	When one class **contains** objects of another class	When one class **is a** type of another class.

Step 4: Describe Relationships between Classes

⊙ **We can describe relationships between classes using relationship arrows offered by UML.**
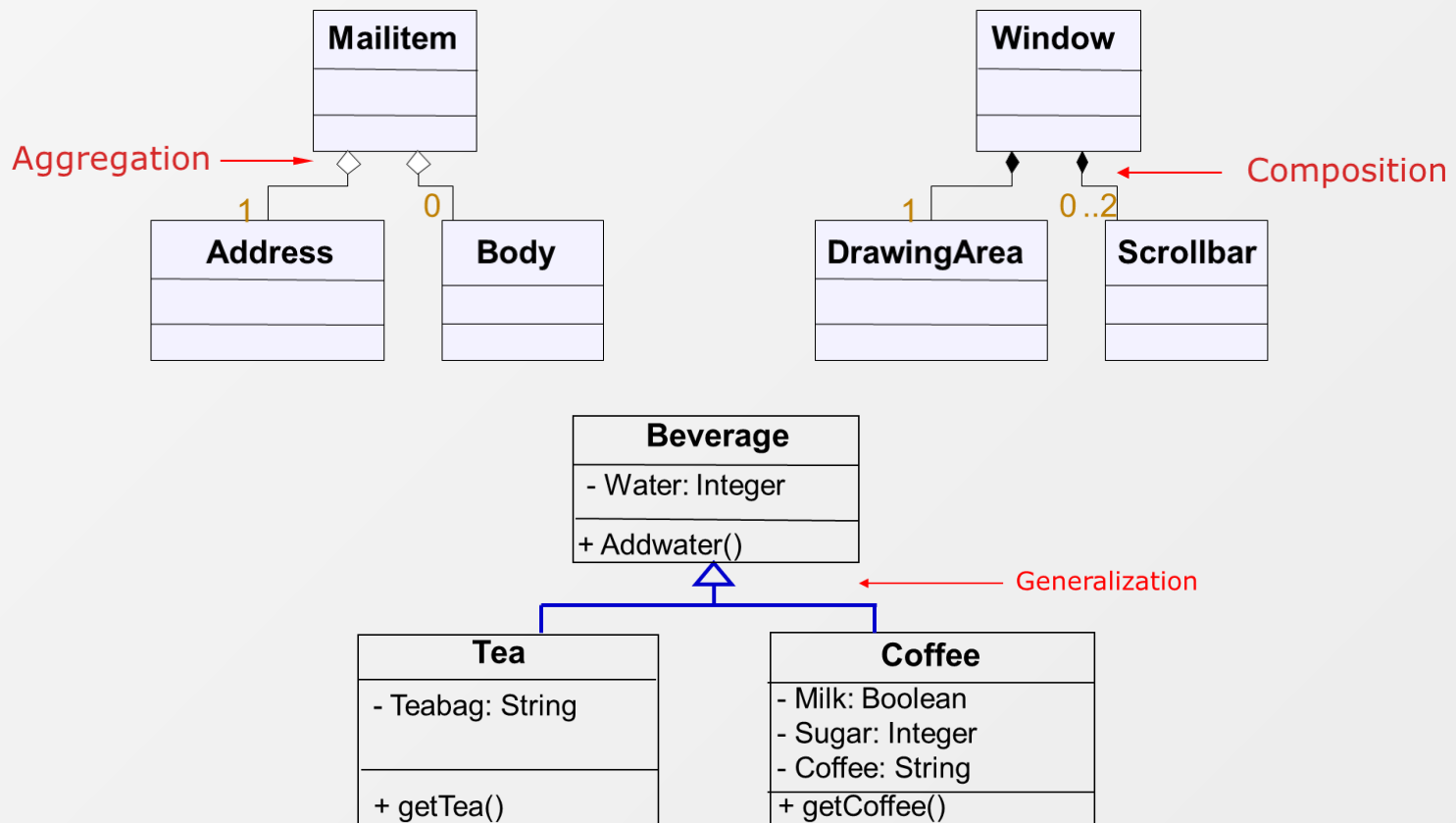
Step 4: Describe Relationships between Classes

○ **We can describe multiplicity of associations by specifying number of instances of one class related to one instance of the other class.**
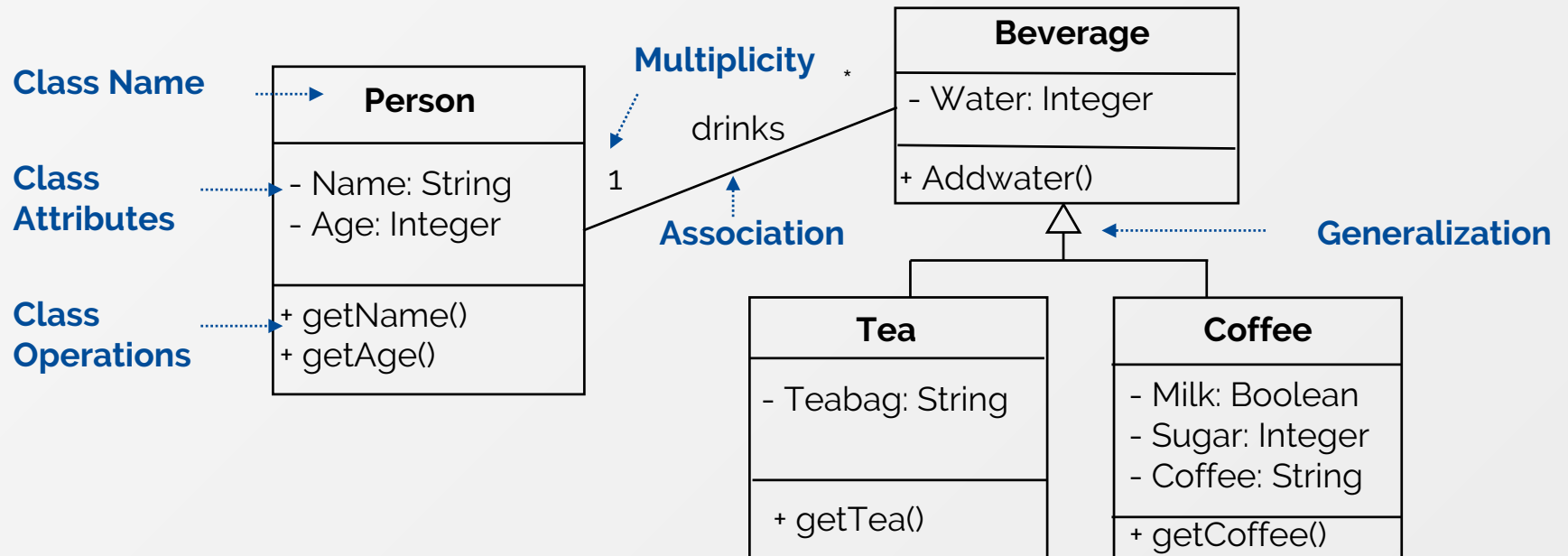
Association name (verb phrase)

Multiplicity   **Multiplicity**

Person
-Name: String -Age: Integer
+getName() +getAge()

1   drinks   *

Coffee
-Milk: Boolean -Sugar: Integer -Coffee: String
+getCoffee()

One person drinks **zero to many** (cup of) coffee

Class Diagram Summary

○ Example class diagram

Class Name ┈┈┈➤ **Person**

Multiplicity

Beverage

- Water: Integer

*

Class Attributes ┈┈┈➤ - Name: String
- Age: Integer

drinks

1

+ Addwater()

Association

Generalization

Class Operations ┈┈┈➤ + getName()
+ getAge()

Tea

- Teabag: String

+ getTea()

Coffee

- Milk: Boolean
- Sugar: Integer
- Coffee: String

+ getCoffee()

UML Diagrams (3/3)

Sequence Diagram
Modeling Ordered Interactions

SE LAB
KAIST

Ordered Interaction between Objects

⊙ **Using sequence diagrams, we can show the sequence of communications between objects in time order.**

- To show sequence of messages exchanged by objects/actors performing a task

- To emphasize time ordering of messages

- To illustrate dynamic view (e.g., scenario, protocol) of a system

Steps for Interaction Modeling

I. **Define and place participants**

- Analyze possible participants in a use case
- Place participants with lifeline

II. **Describe messages**

- Define messages
- Describe message passing with activation bar & arrows

III. **Describe creation and destruction of participants**

IV. **Manage complex interactions with sequence fragements**

# Step 1: Define and Place Participants

○ Definition of participants

- Participants are the parts of your system that interact with each other during the sequence.
    - ▸ A sequence diagram is made up of a collection of participants

○ Drawing lifelines for individual participants

- A lifeline states that the part exists at that point in the sequence.
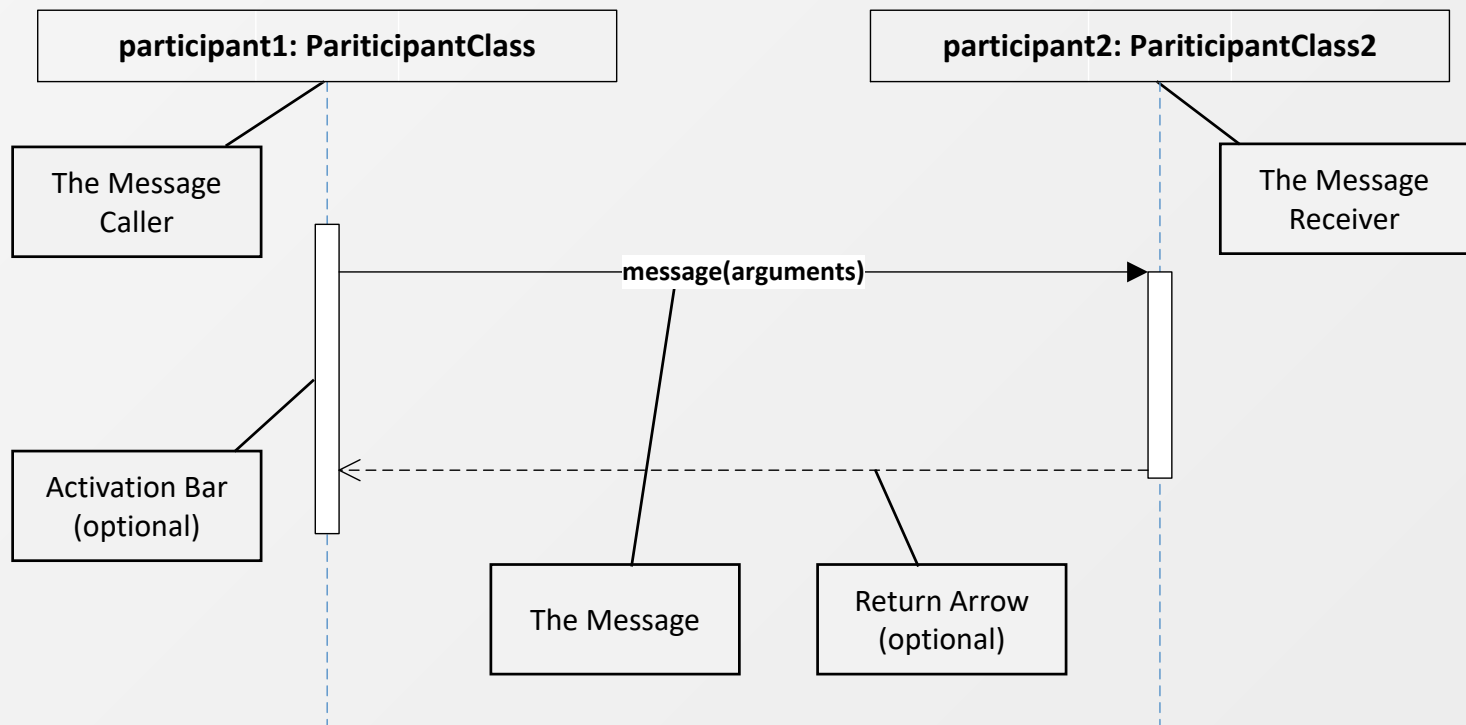
○ Representing active state of a participant

- Activation bars are used to show that a participant is active.

Participant
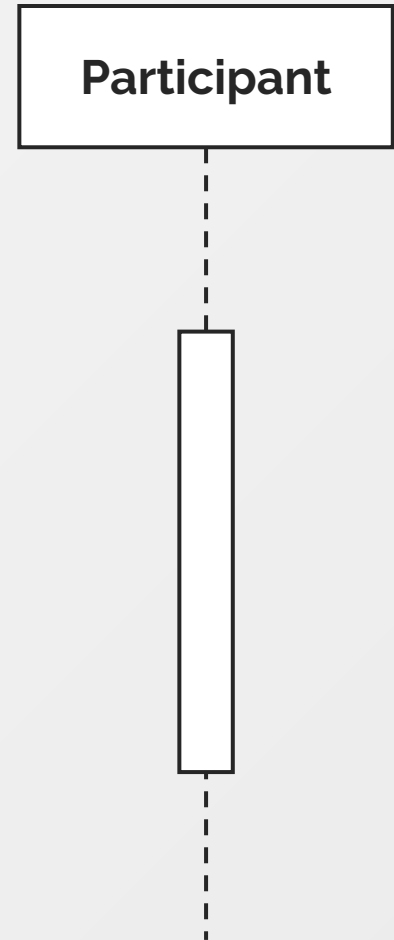
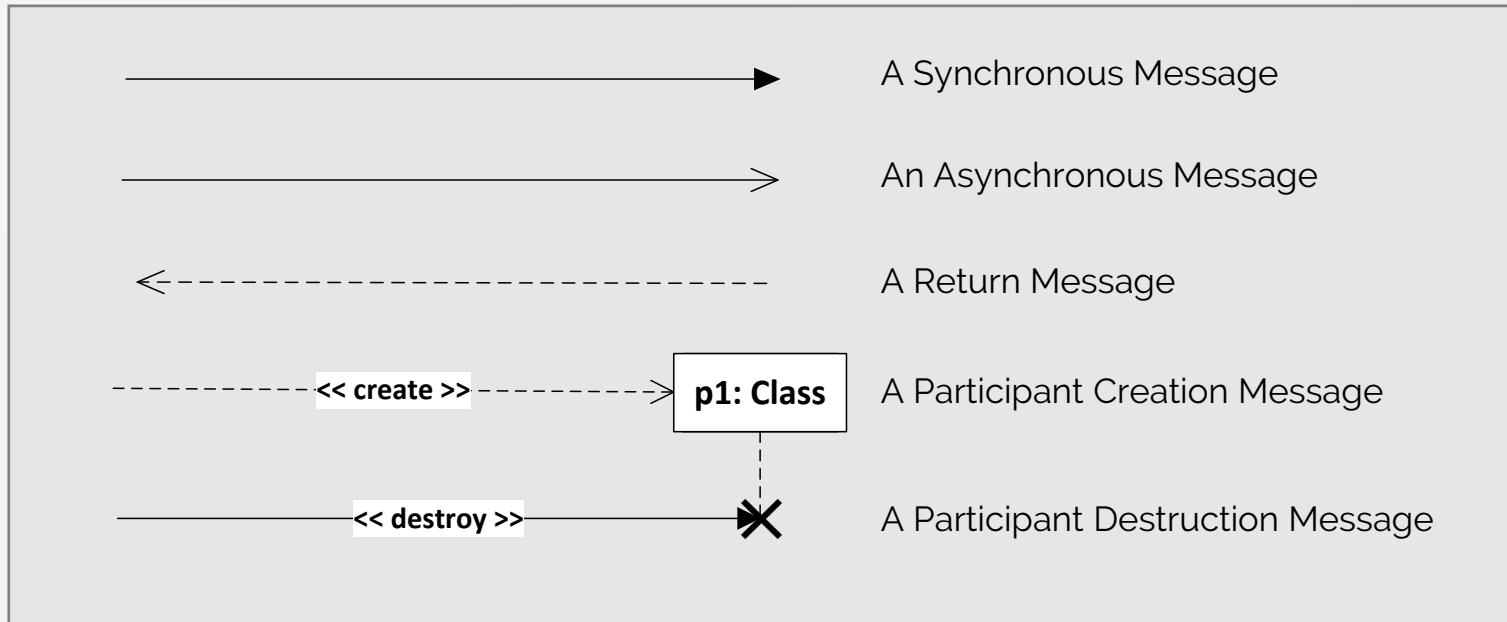# Step 2: Describe Interaction Messages (1/3)

- **Messages are specified using arrows from the message caller to the message receiver.**
  - Whatever direction: left->right, right->left, back to itself

Step 2: Describe Interaction Messages (2/3)

○ **You can specify formats and types of messages.**

- Format of a message signature:
  - ▸ `attribute = signal_or_message_name (arguments) : return_type`
- Five types of message arrow:

|	
──────────▶	A Synchronous Message
──────────▷	An Asynchronous Message
◁┄┄┄┄┄┄┄┄┄	A Return Message
┄┄┄ << create >> ┄┄▷ **p1: Class**	A Participant Creation Message
──── << destroy >> ──✕	A Participant Destruction Message

Step 2: Describe Interaction Messages (3/3)

⊙ You can describe nested messages
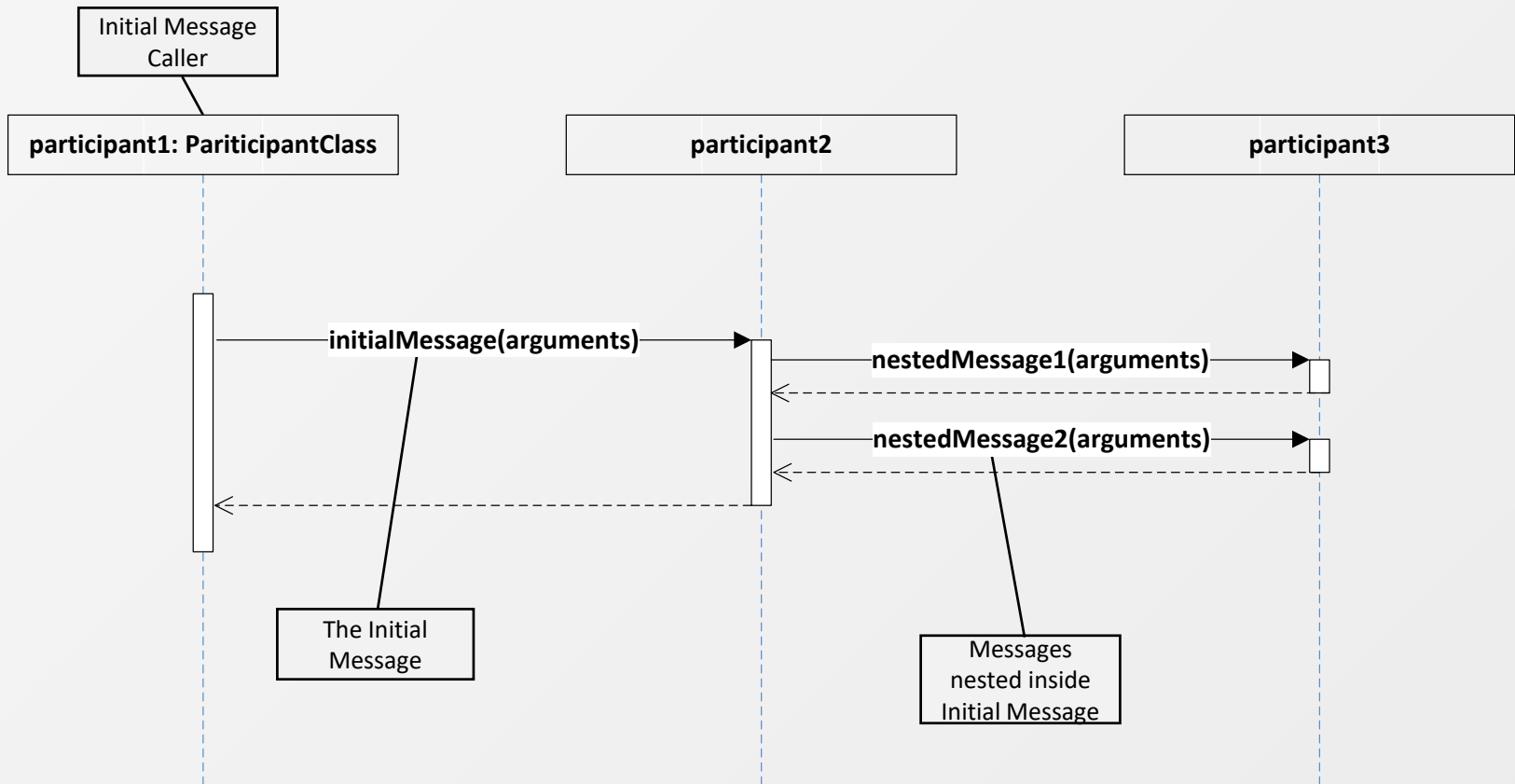
Step 3: Describe Creation & Destruction

- **Participants can be created and destroyed according to the messages that are being passed.**
  - **Creation**: Pass a create(..) message to the participant's lifeline or use the dropped participant box notation
  - **Destruction**: Pass a destroy message to the participant's lifeline and end the participant's lifeline with the deletion cross
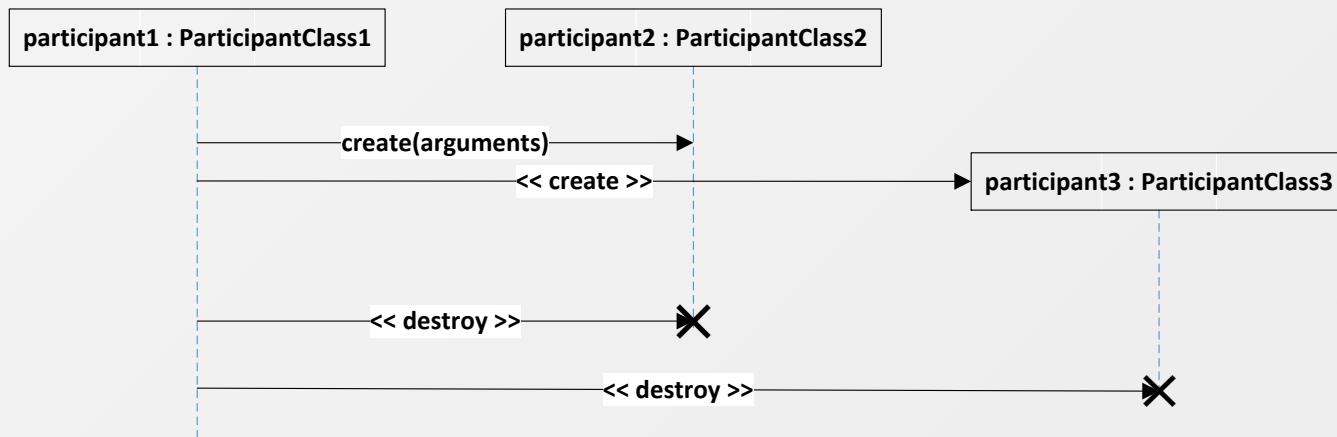
Step 4: Manage Complex Interactions

○ **Using sequence fragments, we can manage complexity in the sequence diagram**

- Sequence fragment: **ref** (referencing)
    - ▸ To reuse already existing sequence diagrams
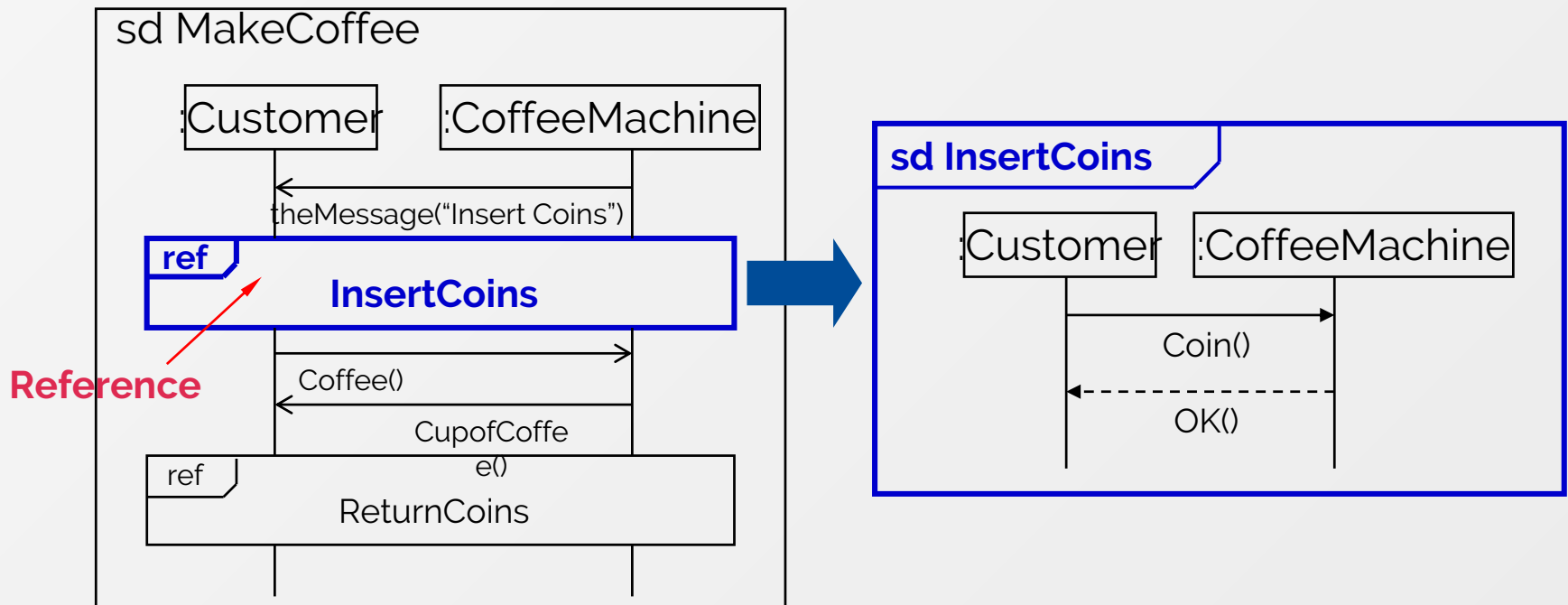
Step 4: Manage Complex Interactions

- **Using sequence fragments, we can manage complexity in the sequence diagram**
  - Sequence fragment: **alt** (alternative)
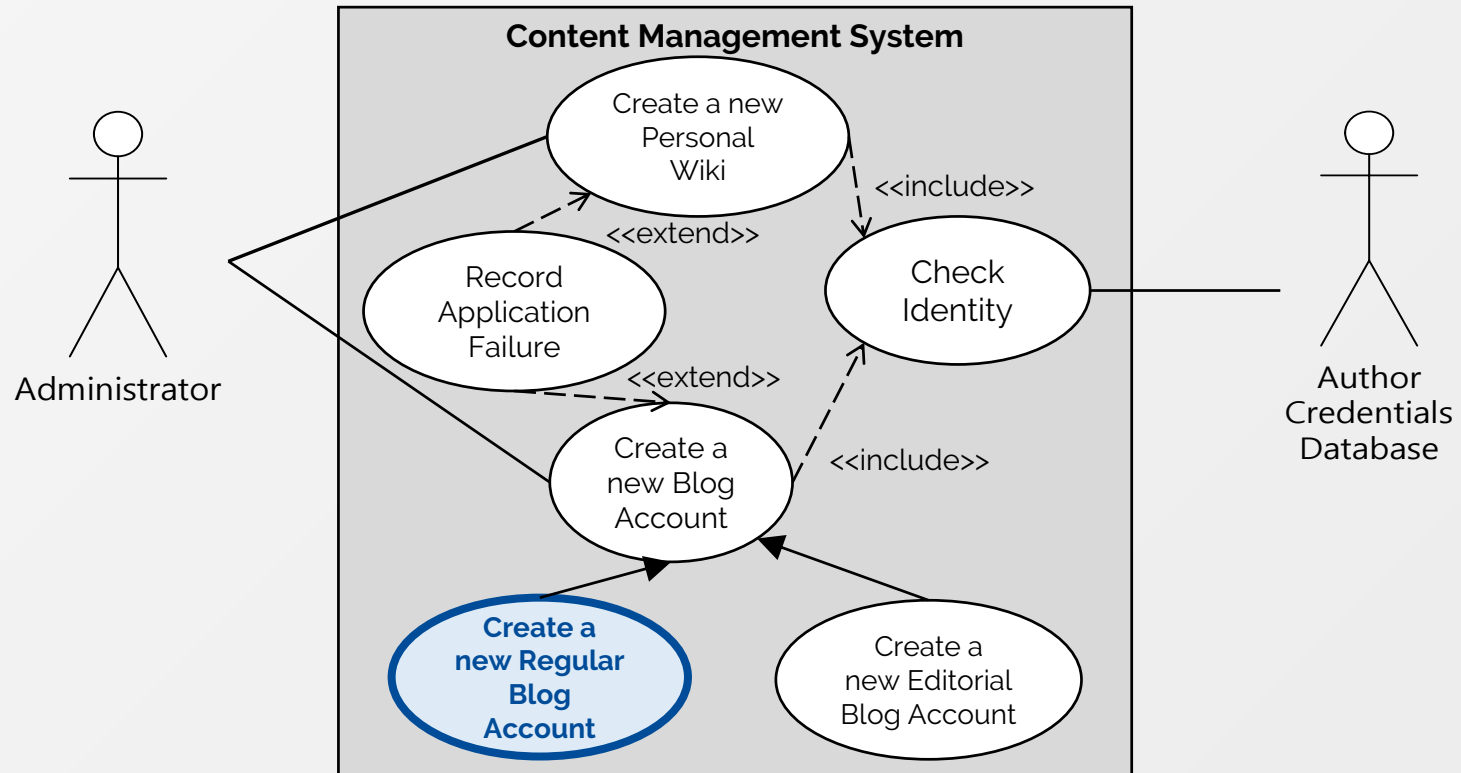   - ▸ To show several alternative interactions (if-else if-else)
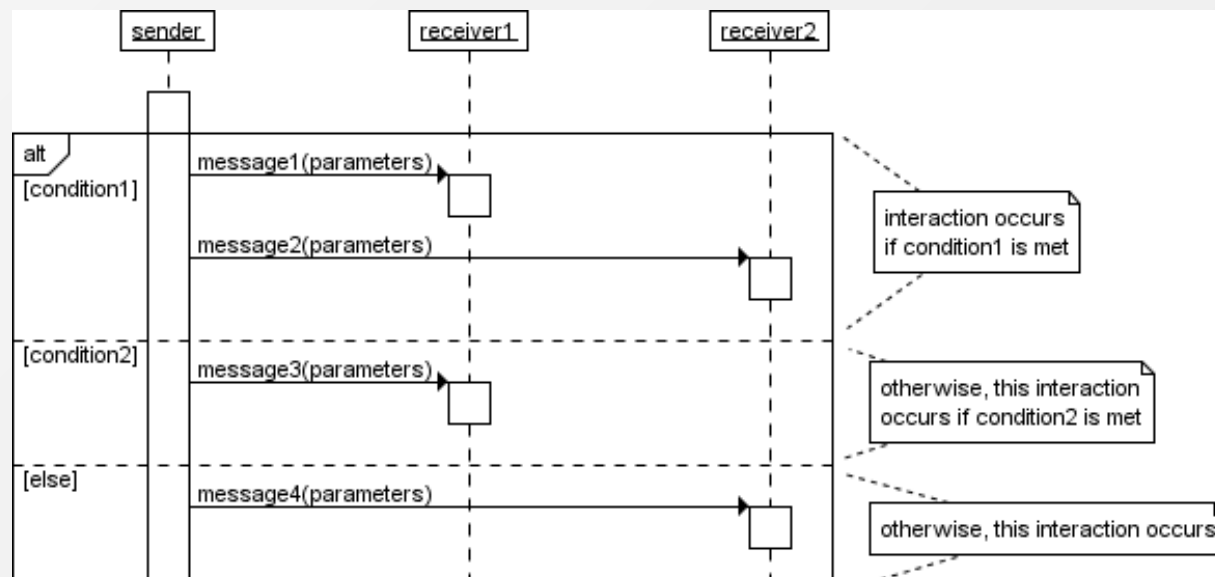
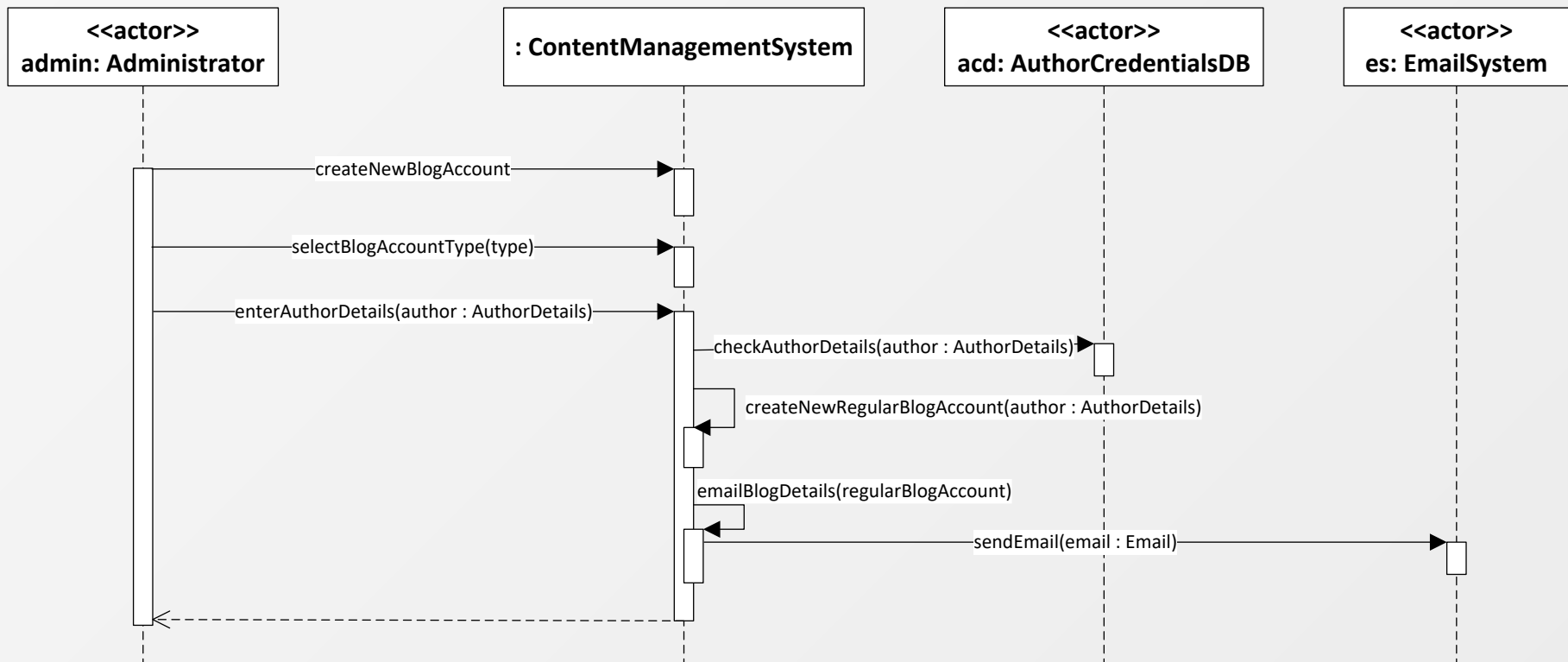More types of sequence fragments? **See appendix**

Example Sequence Diagram

- **In Content Management System (CMS) example, we can model the interactions needed for <Create a new Regular Blog Account> use case**
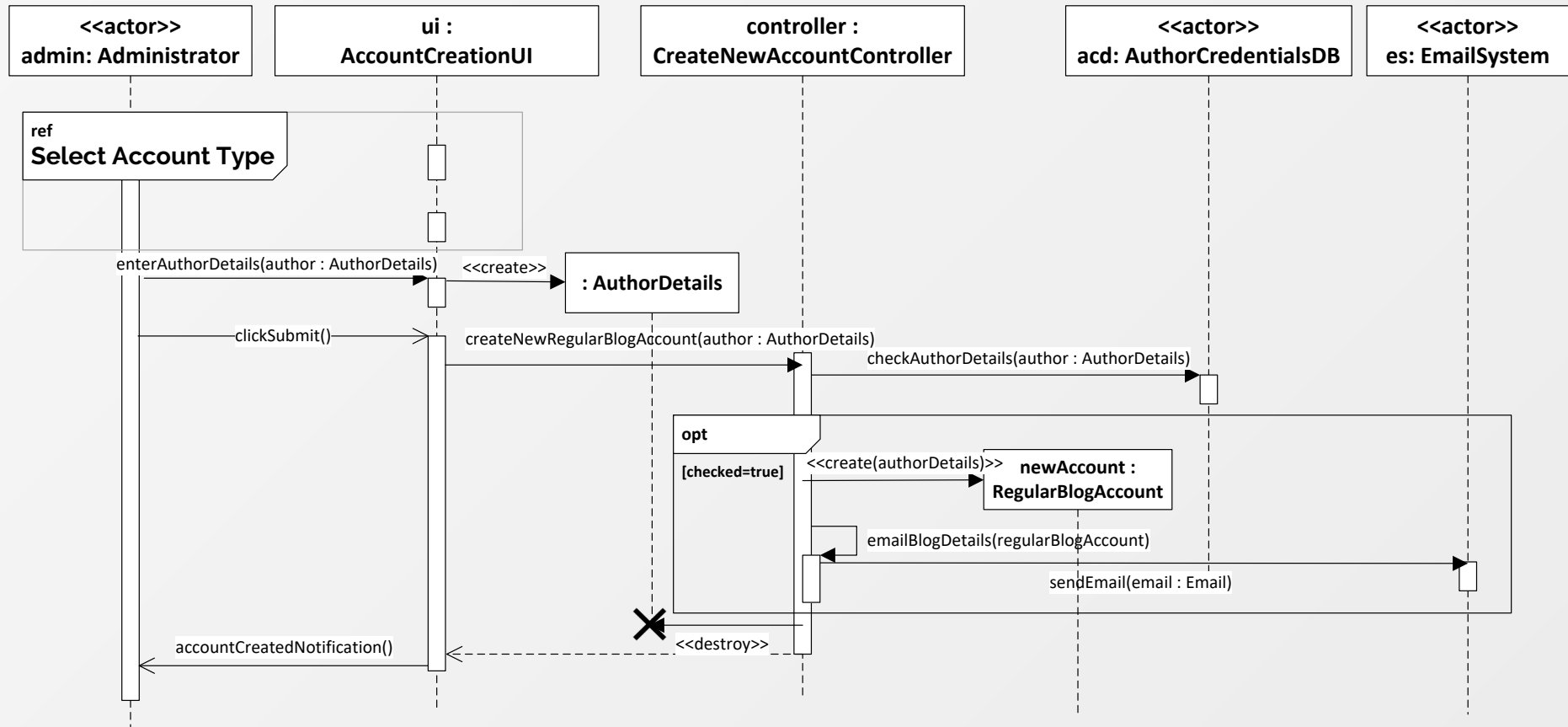
Example Sequence Diagram

○ **Top-level sequence diagram**

Example Sequence Diagram

○ Sequence diagram in detail

UML Support Tools

SE LAB KAIST
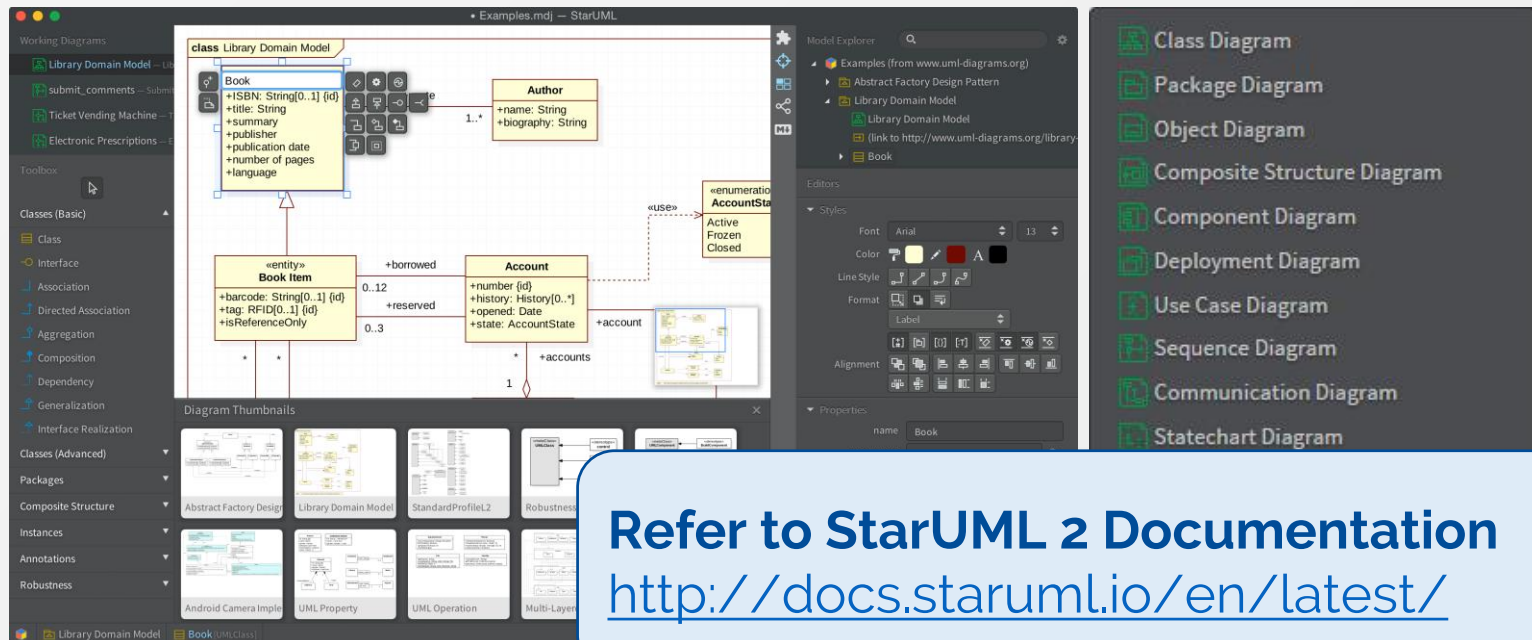
StarUML (UML Tool)

- **A sophisticated software modeling tool**
  - Compatible with UML 2.x
  - Supports 11 key diagrams in UML:
    Class, Use Case, Sequence, Object, Activity, Component, and etc.

Refer to StarUML 2 Documentation
http://docs.staruml.io/en/latest/

StarUML (UML Tool)

Tool information

- Site: http://staruml.io/download
- Latest version: v2.8.0 (3/20/2017)
- Supporting various OSs: Windows, macOS, Linux
- Unlimited use for evaluation

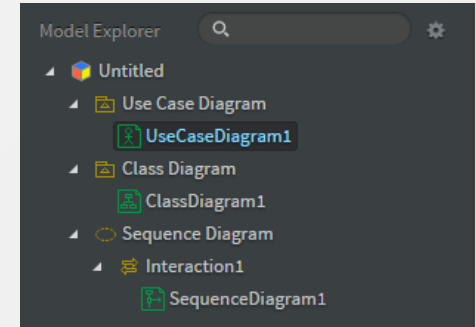**You may start a UML project with:**

- An empty project consisting of
  - ‣ Use Case diagram
  - ‣ Class diagram
  - ‣ Sequence diagram
- The empty project file can be downloaded from KLMS or SE course site.

http://staruml.io/
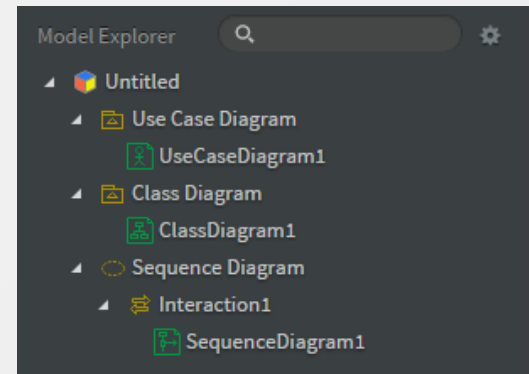
How to draw a Diagram?

1. **Select a diagram from Model Explorer panel.**
   - You can add or delete models.
   - Even you can change some properties including name, in **Editors** panel.

2. **Select a figure you want to draw.**
   - You can find a lot of figures in **Toolbox** panel.
   - Click on a figure, then draw it on the canvas by making a drag&drop.
   - If you click one more, the figure is locked so that you can draw multiple figures in a row.
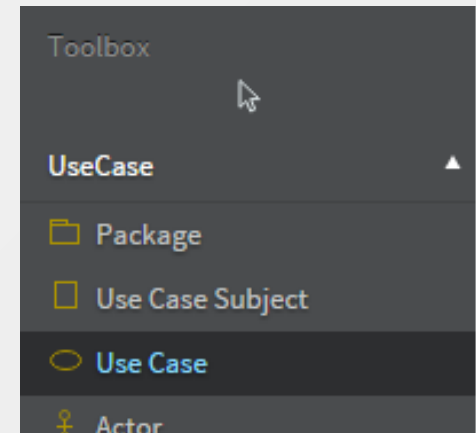
How to draw a Diagram?

3. **Set proper values for the figure.**
- Change its visibility and name.
- A note can be added for the figure to describe detailed information.
- You can easily add some relevant figures by clicking buttons on the right.
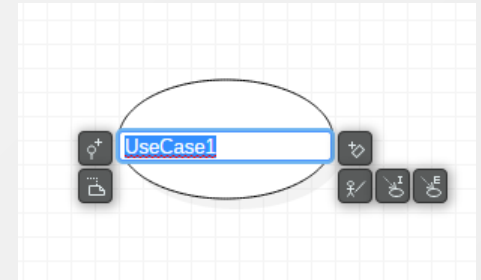
4. **Set relationship between figures.**
- As seen at step 2, select a relation figure.
- Drag from one figure to another figure.
- Now you have a relation between figures, and set proper values for the relation.
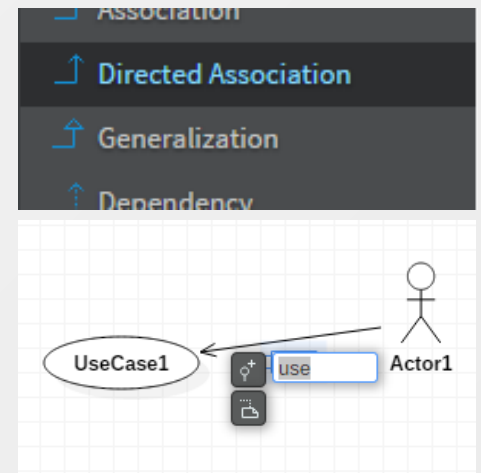
How to draw a Diagram?

5. **Check your figures.**
- You can see the hierarchy of the project in **Model Explorer** panel.
- It includes your diagrams and figures.
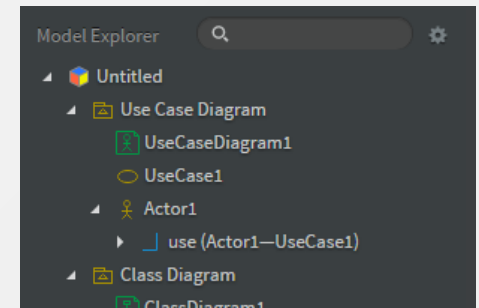
6. **Reuse the figures.**
- If you want to use the same figure in the same diagram or in another diagram, it is simply done by dragging it from **Model Explorer** panel to the canvas.
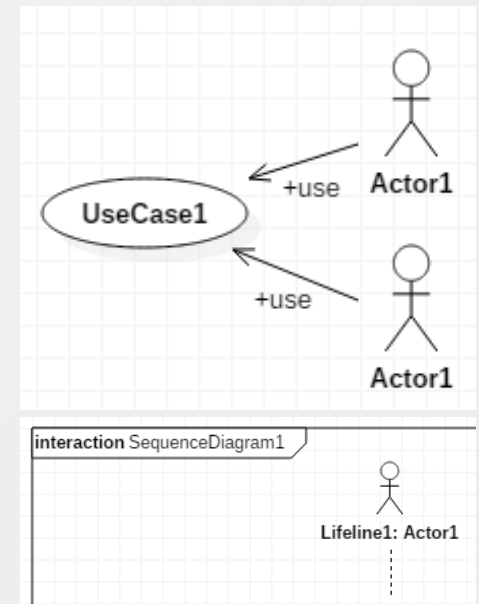
Star UML Guide (in both English & Korean)

- **English** http://staruml.sourceforge.net/docs/user-guide(en)/toc.html
  - Modeling Use Case Diagram
    - ▸ http://staruml.sourceforge.net/docs/user-guide(en)/ch05_1.html
  - Modeling Class Diagram
    - ▸ http://staruml.sourceforge.net/docs/user-guide(en)/ch05_2.html
  - Modeling Sequence Diagram
    - ▸ http://staruml.sourceforge.net/docs/user-guide(en)/ch05_3.html

- **Korean** http://staruml.sourceforge.net/docs/user-guide(ko)/toc.html
  - Modeling Use Case Diagram
    - ▸ http://staruml.sourceforge.net/docs/user-guide(ko)/ch05_1.html
  - Modeling Class Diagram
    - ▸ http://staruml.sourceforge.net/docs/user-guide(ko)/ch05_2.html
  - Modeling Sequence Diagram
    - ▸ http://staruml.sourceforge.net/docs/user-guide(ko)/ch05_3.html

Other UML Support Tools

- **List of Unified Modeling Language tools**
  - https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools

Name	Creator	Platform / OS	First public release	Latest stable release	Open source	Software license	Programming language used
ArgoUML	Tigris.org	Cross-platform (Java)	1998-04	2011-12-15[1]	Yes	EPL	Java, C++ (as module)
Astah	Change Vision, Inc.	Windows, macOS, Linux	2009-10-19	2016-09-28	No	Commercial, Free trial, Free edition (Community version)	Java, C++, C#, PHP
ATL	Obeo, INRIA / Free software community	Cross-platform (Java)	Unknown	2010-06-23	Yes	EPL	Java
Borland Together	Borland	Cross-platform (Java)	Unknown	2008	No	Commercial	Unknown
BOUML	Bruno Pagès	Cross-platform	2005-02-26	2016-06-04	No	Commercial starting from v5.0,[2] GPL before v5.0	C++/Qt and Java ("plug-out")
CaseComplete	Serlio Software	Windows	2004	2013-04	No	Commercial	C#
ConceptDraw PRO	CS Odessa	Windows, macOS	1993	2010 (v9)	No	Commercial	Unknown
Dia	Alexander Larsson/GNOME Office	Cross-platform (GTK+)	2004?	2012-07-05	Yes	GPL	C
Eclipse UML2 Tools[3]	Eclipse Foundation	Cross-platform (Java)	2007	2009-06-19	Yes	EPL?	Java
Edraw Max	Edrawsoft	Windows, Linux, macOS	2004	2015-03	No	Commercial	C++
Enterprise Architect	Sparx Systems	Windows (supports Linux and macOS installation)	2000	2015-06-18	No	Commercial	C++
Gliffy	Gliffy	Chrome, Safari, Firefox, Internet Explorer 9+	2006-08-01	2015-01 (v. 5.1)	No	Commercial, Free trial	HTML5 and JavaScript
Lucidchart	Lucid Software	Windows, macOS, Linux, Solaris	2008-12	2014-10-07	No	Commercial / Free (educational)	HTML5 and JavaScript
MagicDraw	No Magic	Cross-platform (Java)	1998	2014-11-17 (v18.1)	No	Commercial	Java
Microsoft							
Microsoft							
Microsoft Managem							
Modelio							
MyEclipse							
MySQL W							
NClass	Balazs Tihanyi	Windows, macOS, Linux, Unix	2006-10-15	2011-06-06	Yes	GPL	C#
NetBeans[5]	Oracle Corporation	Windows, macOS, Linux, Unix	1996	2013-02-21	Yes	CDDL or GPL2	Java

> Choose a **freeware** tool that **supports UML 2.0** as possible.
> But, you can use most of tools **with a free trial** :)

http://staruml.io/

Thank You.

2018 Fall - CS350: Introduction to Software Engineering
Unified Modeling Language (UML)

Yong-Jun Shin, Doo-Hwan Bae
ppt slides by Young-Min Baek

Appendix

Software Requirements Specification (SRS)

- **Definition**
  - A software requirements specification (SRS) is a comprehensive description of the intended purpose and environment for software under development.
  - It lays out functional and non-functional requirements and may include a set of use cases that describe user interactions that the software must provide.

- **Benefits**
  - Minimizing the time and cost: A SRS is a communication tool between stakeholders and software designers. It permits a rigorous assessment of requirements before design can begin and reduces later redesign.
  - Planning ahead: A good SRS defines how an application will interact with system hardware, other programs and users in a wide variety of real-world situations.

- **Goals**
  - Facilitating reviews
  - Describing the scope of work
  - Providing a reference to software designers (i.e. navigation aids, document structure)
  - Providing a framework for testing primary and secondary use cases

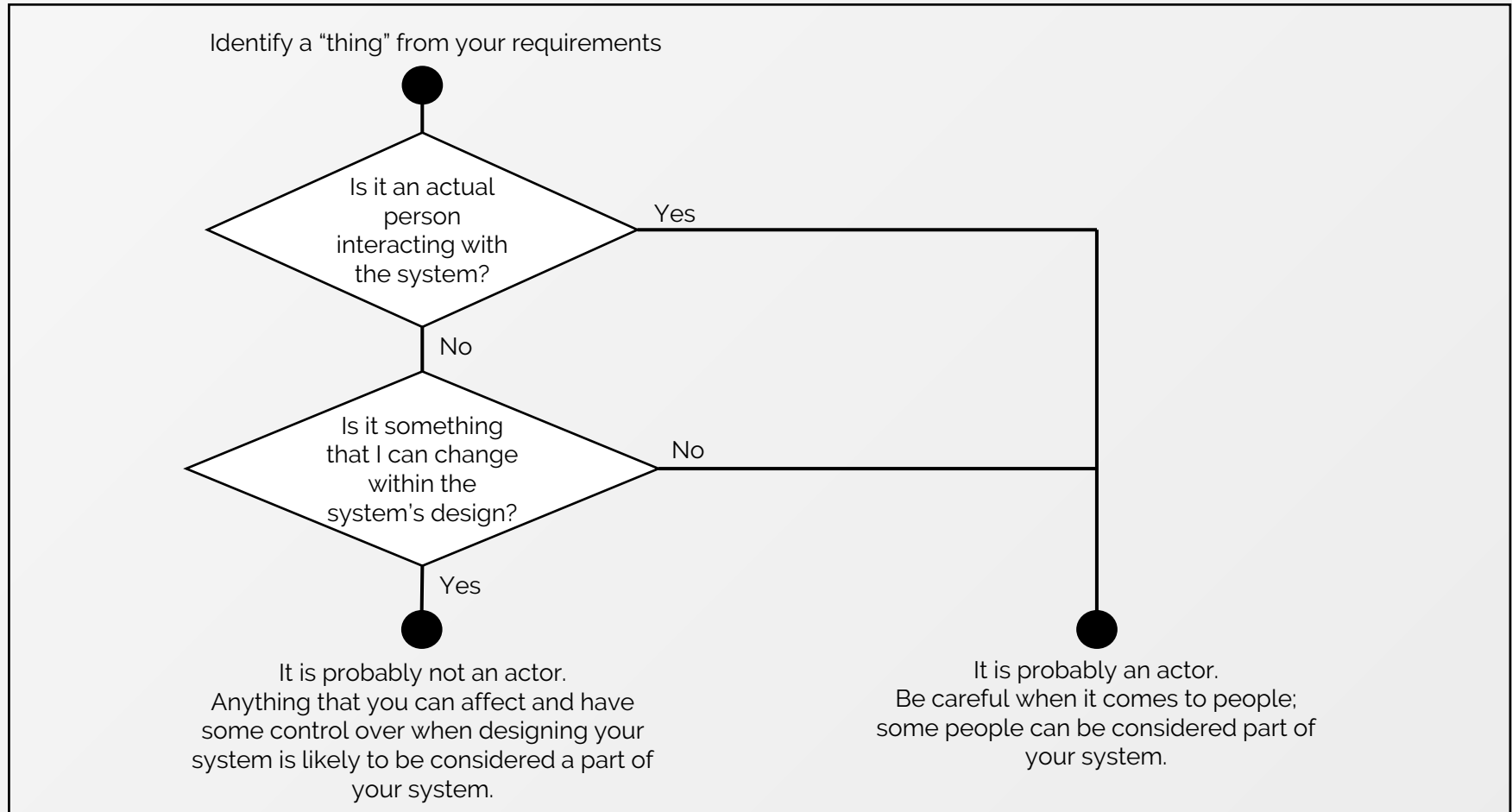# Requirements: Functional & Non-functional Requirements

- **Functional Requirements**
  - Functional requirements are quantities that specify the performance of a design.
  - They are related to the functions or behaviors of the design.
  - What does the system do?
    - i.e. a functional requirement for a milk carton would be "ability to contain fluid without leaking"

- **Non-functional Requirements**
  - Non-functional requirements specify characteristics of the design that are not performance based.
  - They specify how the system should behave and are a constraint upon the systems behavior.
  - They include quantities of a system, such as capacity, usability, security, maintainability, reliability, availability and so on.
    - i.e. A non-functional requirement for a hard hat might be "must not break under pressure of less than 10,000 PSI"

Identifying an actor

Identify a "thing" from your requirements

⬤

Is it an actual person interacting with the system? — **Yes**

No

Is it something that I can change within the system's design? — **No**

Yes

⬤

It is probably not an actor.
Anything that you can affect and have some control over when designing your system is likely to be considered a part of your system.

⬤

It is probably an actor.
Be careful when it comes to people; some people can be considered part of your system.

Stakeholders (1/2)

⊙ **A stakeholder in the architecture of a system is an individual, team, organization, or classes thereof, having an interest in the realization of the system[*].**

• The architect must ensure that there is adequate stakeholder representation across the broad, including nontechnology stakeholders (e.g., acquirers and users) and technology-focused ones (developers, system administrators, and maintainers)

Acquirers	Oversee the procurement of the system or product
Assessors	Oversee the system's conformance to standards and legal regulation
Communicators	Explain the system to other stakeholders via its documentation and training materials
Developers	Construct and deploy the system from specifications (or lead the teams that do this)
Maintainers	Manage the evolution of the system once it is operational
Production Engineers	Design, deploy, and manage the hardware and software environments in which the system will be built, tested, and run
Suppliers	Build and/or supply the hardware, software, or infrastructure on which the system will run
Support Staff	Provide support to users for the product or system when it is running
System Administrators	Run the system once it has been deployed
Testers	Test the system to ensure that it is suitable for use
Users	Define the system's functionality and ultimately make use of it

Stakeholders (2/2)

- **A stakeholder in an organization is (by definition) any group or individual who can affect or is affected by the achievement of the organization's objectives**[1,2].

- **Stakeholders are participants <in the development process> together with any other individuals, groups or organizations whose actions can influence or be influenced by the development and use of the system whether directly or indirectly**[3].

- **The people and organizations affected by the application**[4].

- **Stakeholders are people who have a stake or interest in the project**[5].

- **A stakeholder is anyone whose jobs will be altered, who supplies or gains information from it, or whose power or influence within the organization will increase and decrease**[6].

[1] Helen Sharp, Anthony Finkelstein, and Galal Galal, "Stakeholder Identification in the Requirements Engineering Process," Database and Expert Systems Applications, 1999.
[2] Freeman, R.E. (1984) Strategic Management: A stakeholder approach, Pitman, Boston.
[3] Pouloudi, A. (1997) 'Stakeholder Analysis as a FrontEnd to Knowledge Elicitation', AI & Society, 11, 122–137.
[4] Conger, S. (1994) The New Software Engineering, International Thomson Publishing
[5] Cotterell, M. and Hughes, B. (1995) Software Project Management, International Thomson Publishing.
[6] Dix, A., Finlay, J. Abowd, G. and Beale, R. (1993) HumanComputer Interaction, Prentice-Hall.
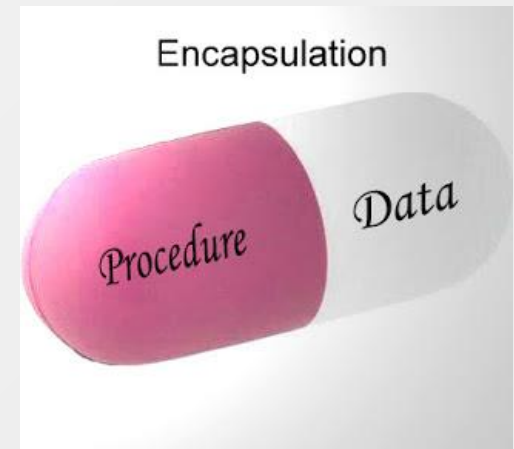
SE LAB KAIST

Abstraction & Encapsulation

- **Abstraction**
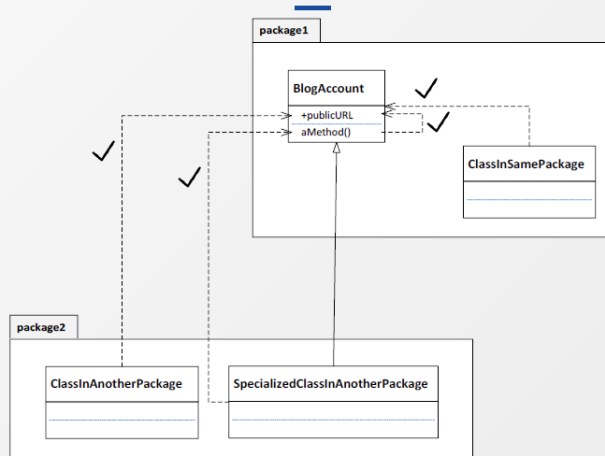 - Discarding irrelevant details within a given context

- **Encapsulation**[1] (same as encapsulation of OO)
  - Enabling a class to hide the inner details of how it works from the outside world and only expose the operation and data that it chooses to make accessible.
    - ▸ Information hiding concept
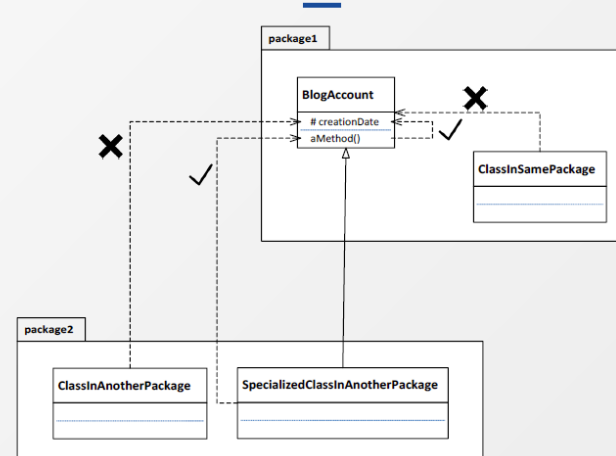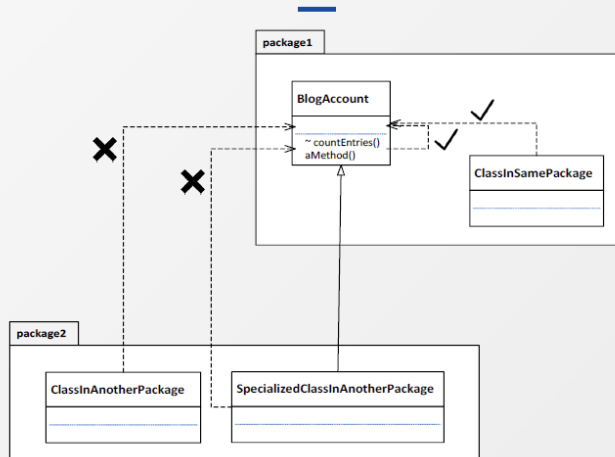 - Keeping the data safe and secure from external interventions

Encapsulation

Procedure   Data

[1] http://marchoeijmans.blogspot.kr/2012/06/encapsulation-concept.html

Visibility of Classes

Sequence Fragments

Type	Parameters	Why is it useful?
ref	None	Represents an interaction that is defined elsewhere in the model. Helps you manage a large diagram by splitting, and potentially reusing, a collection of interactions. Similar to the reuse modeled when the *<<include>>* use case relationship is applied.
assert	None	Specifies that the interactions contained within the fragment box must occur exactly as they are indicated; otherwise the fragment is declared invalid and an exception should be raised. Works similar fashion to the *assert* statement in Java. Useful when specifying that every step in an interaction must occur successfully.
loop	min times, max times, [guard_condition]	Loops through the interactions contained within the fragment a specified number of times until the guard condition is evaluated to false. Very similar to the Java and C# *for(..)* loop. Useful when you are trying execute a set of interactions a specific number of times.
break	None	If the interactions contained within the break fragment occur, then any enclosing interaction, most commonly a loop fragment, should be exited. Similar to the *break* statement in Java and C#.
alt	[guard_condition1] ... [guard_condition2] ... [else]	Depending on which guard condition evaluates to true first, the corresponding sub-collection of interactions will be executed. Helps you specify that a set of interactions will be executed only under certain conditions. Similar to an *if(..) else* statement in code.

Sequence Fragments

Type	Parameters	Why is it useful?
opt	[guard_condition]	The interactions contained within this fragment will execute only if the guard condition evaluates to true. Similar to a simple *if(..)* statement in code with no corresponding *else*. Especially useful when showing steps that have been reused from another use case's sequence diagrams, where *<<extend>>* is the use case relationship.
neg	None	Declares that the interactions inside this fragment are not to be executed, ever. Helpful if you are just trying to mark a collection of interactions as not executed until you're sure that those interactions can be removed. Most useful if you happen to be lucky enough to be using an Executable UML tool where you sequence diagrams are actually being run. Also can be helpful to show that something cannot be done. Works in similar fashion to commenting out some method calls in code.
par	None	Specifies that interactions within this fragment can happily execute in parallel. This is similar to saying that there is no need for any thread-safe locking required within a set of interactions.
region	None	Interaction within this type of fragment are said to be part of a critical region. A critical region is typically an area where a shared participant is updated. Combined with parallel interactions, specified using the par fragment type, you can model where interactions are not required to be thread- or process-safe (par fragment) and where locks are required to prevent parallel interactions interleaving (region fragment). Has similarities synchronized blocks and object locks in Java.