

CSE201: Monsoon 2020  
Advanced Programming

# **Lecture 15: Unified Modeling Language**

Vivek Kumar

Computer Science and Engineering

IIT Delhi

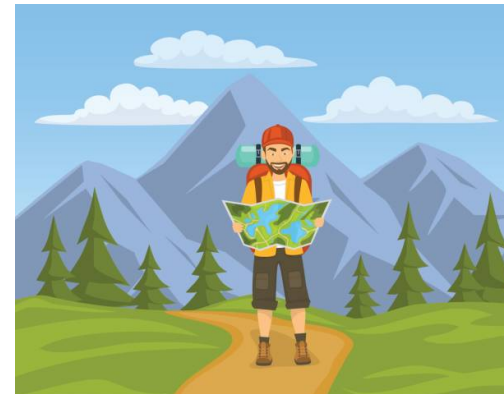
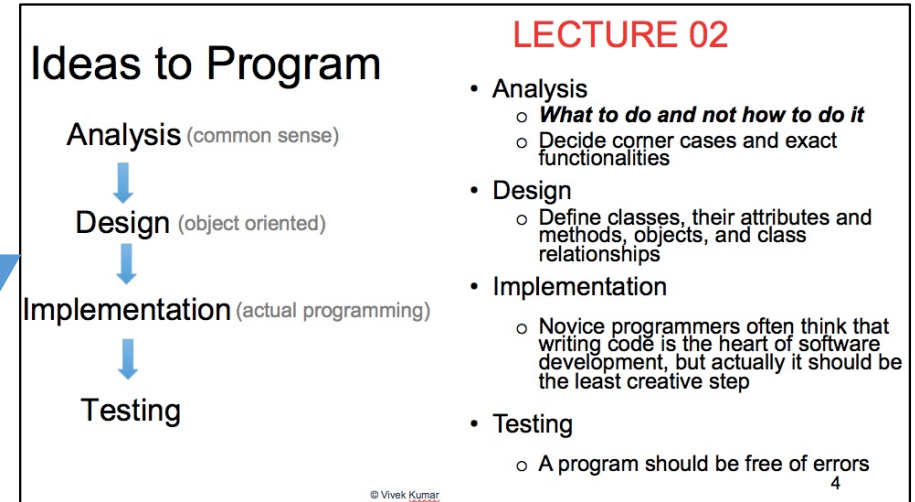
[vivekk@iiitd.ac.in](mailto:vivekk@iiitd.ac.in)

# Today's Lecture

- Introduction to UML
  - We already covered UML in bits and pieces in prior lectures
    - Sequence diagram (Lecture 2)
    - Representing class relationships (Lectures 3–6)
- Relationships in use case diagrams
- Goal of this lecture is to give you more familiarity with UML
  - You can model 80% of problems by using about 20% UML
  - We will only cover less than 20% here
    - Not possible to teach everything...

# What is UML?

- UML stands for Unified Modeling Language
- It's a widely used modeling language in the field of software engineering
- It's used to analyze, design, and implement software-based systems
- Pretty pictures (diagrams) →



# Motivations for UML

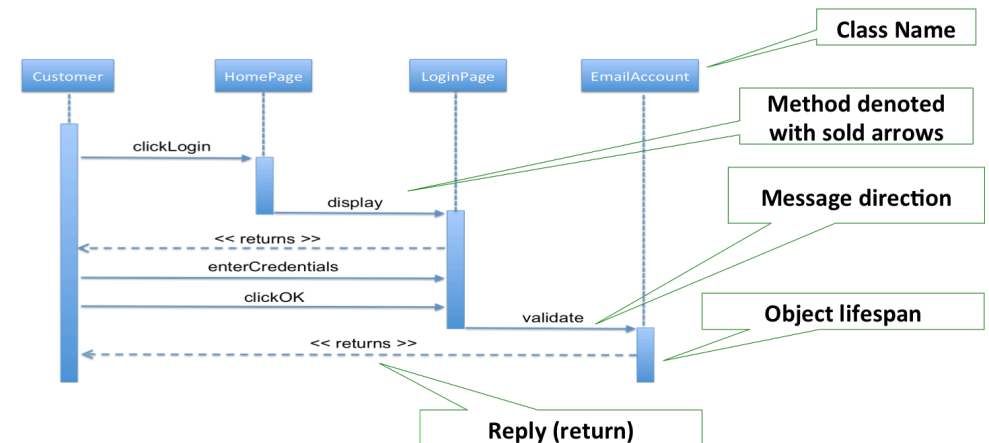
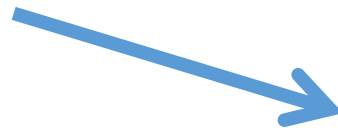
- We need a modeling language to:
  - help develop efficient, effective and correct designs, particularly Object Oriented designs
  - communicate clearly with project stakeholders (concerned parties: developers, customer, etc)
  - give us the “big picture” view of the project

# UML Diagrams

Three types of UML diagrams that we will cover:

1. **Class diagrams:** Represents static structure
2. **Use case diagrams:** Sequence of actions a system performs to yield an observable result to an actor
3. **Sequence diagrams:** Shows how groups of objects interact in some behavior

- Already covered in Lecture 02

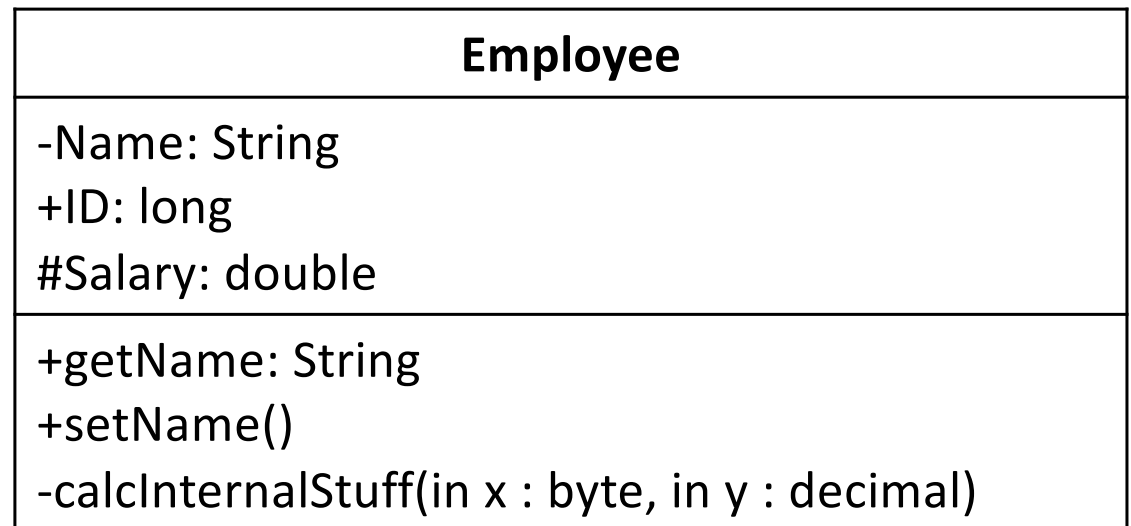


# Class Diagrams

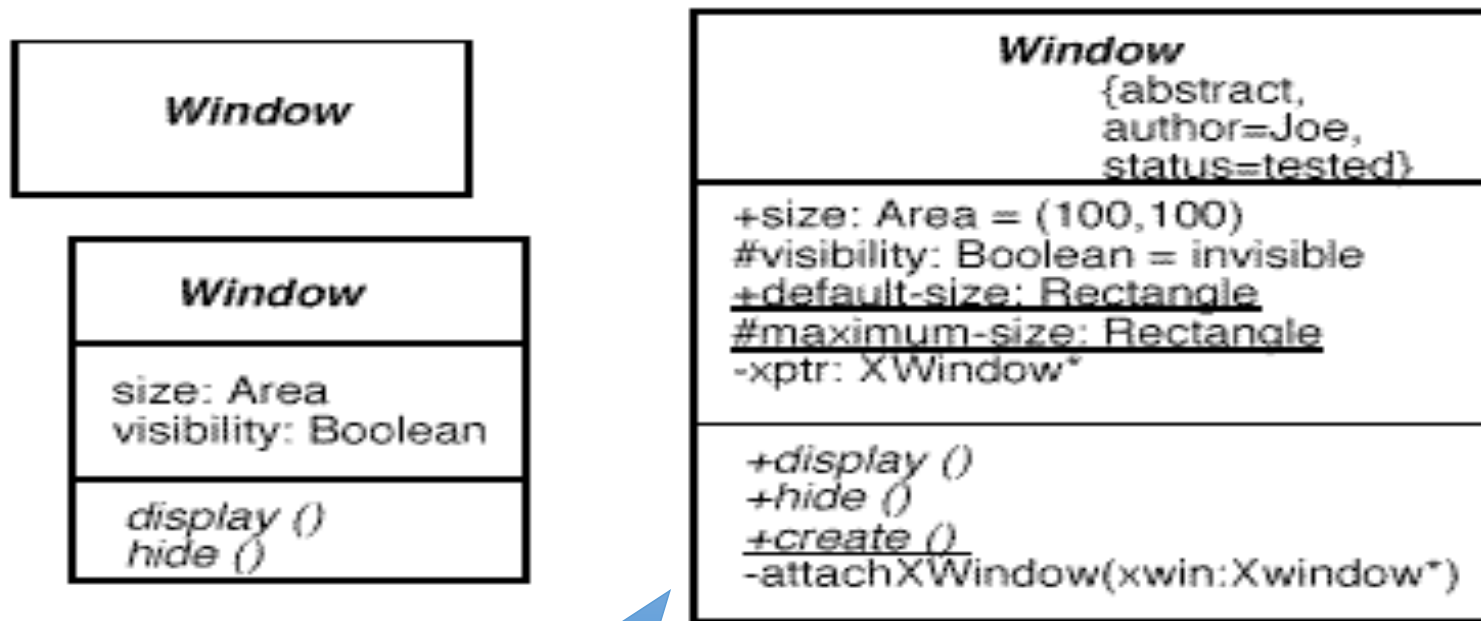
- Better name: “Static structure diagram”
  - Doesn't describe temporal aspects
  - Doesn't describe individual objects: Only the overall structure of the system
- There are “object diagrams” where the boxes represent instances
  - Rarely used and not covered in this course

# 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
- Abstract class/methods
  - Name in italics



# Different Levels of Specifying Classes



Use this for your project



# Class Relationships

- UML diagrams for these class relationships are already covered before (Lectures 04, 05 and 08)
  - Association
  - Composition
  - Dependency
  - Inheritance
- We will only cover binary association relationship here

# Class Relationship: Binary Association

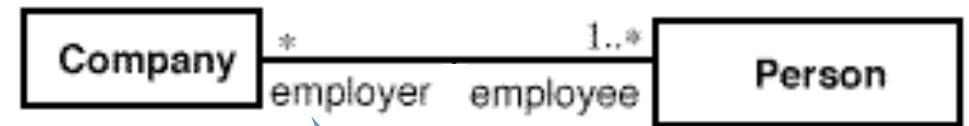
Both entities “Knows About” each other (two-way association)



# UML Multiplicities

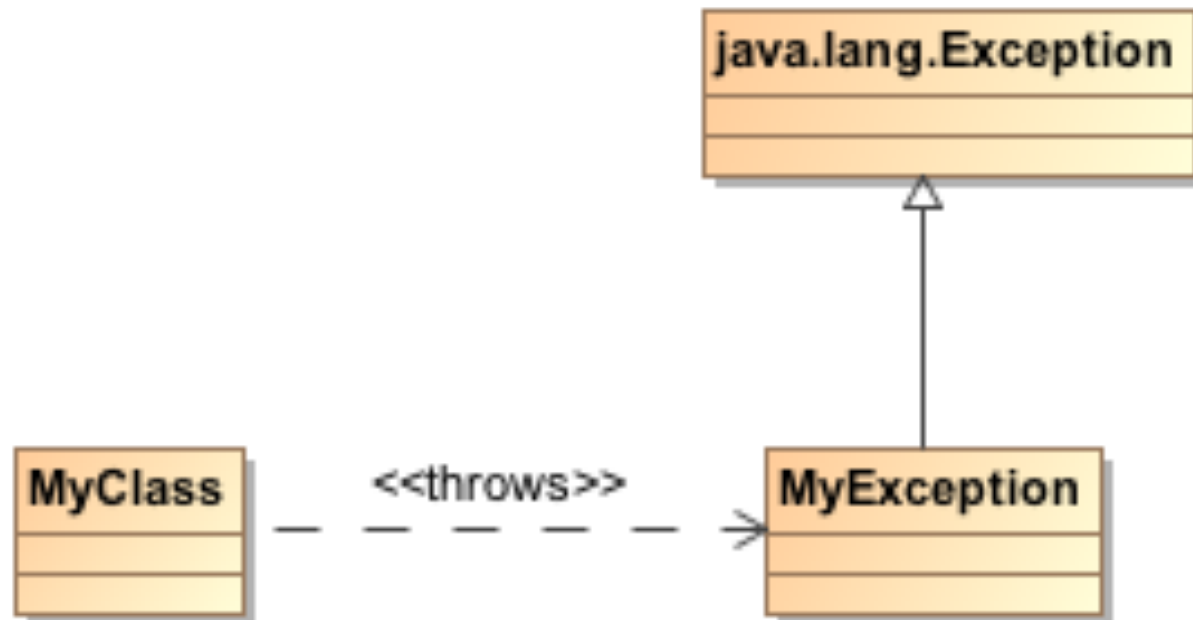
Links on associations to specify more details about the relationship

Multiplicities	Meaning
0..1	zero or one instance. The notation " <i>n</i> .. <i>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

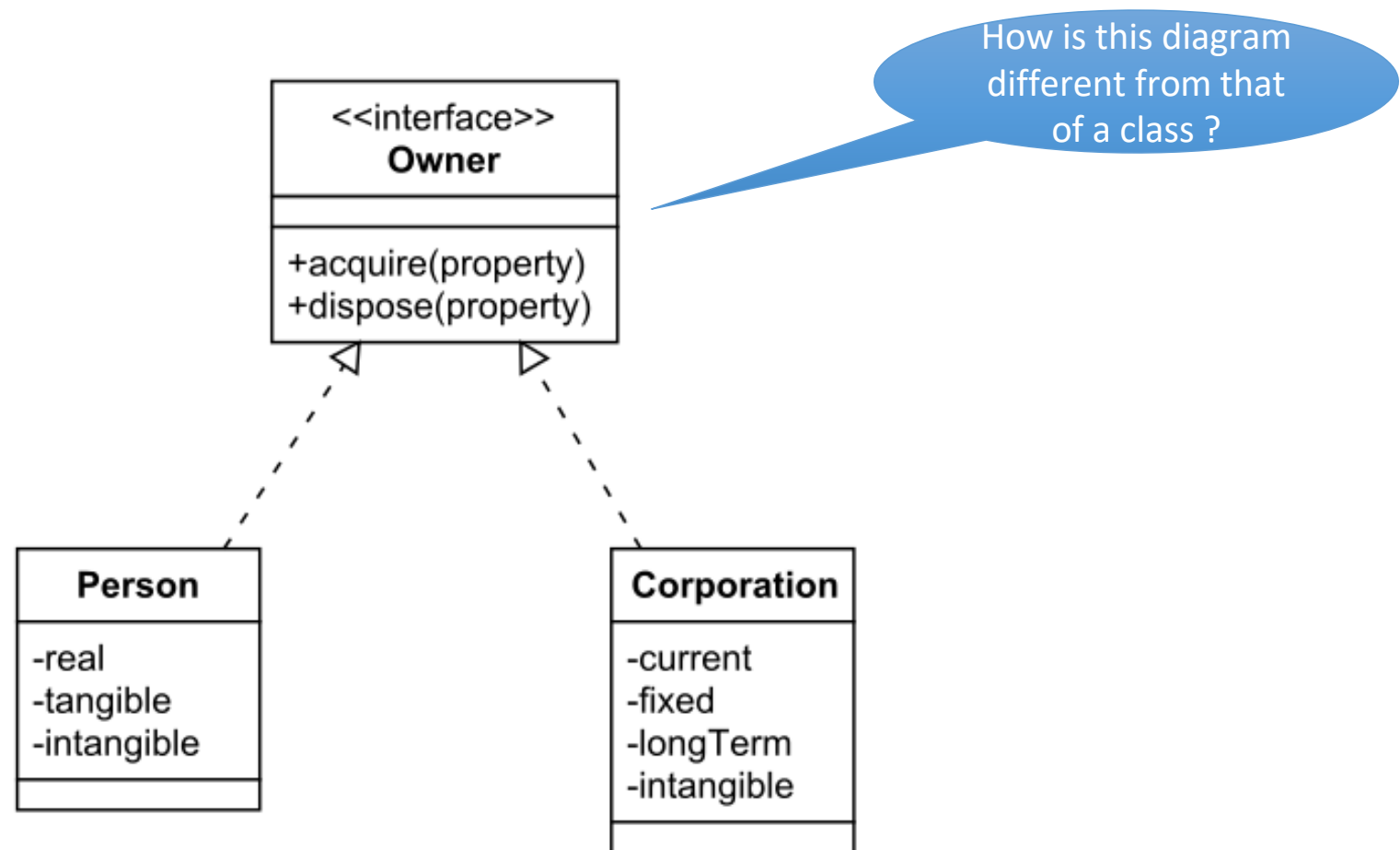


How you will implement?

# Exceptions

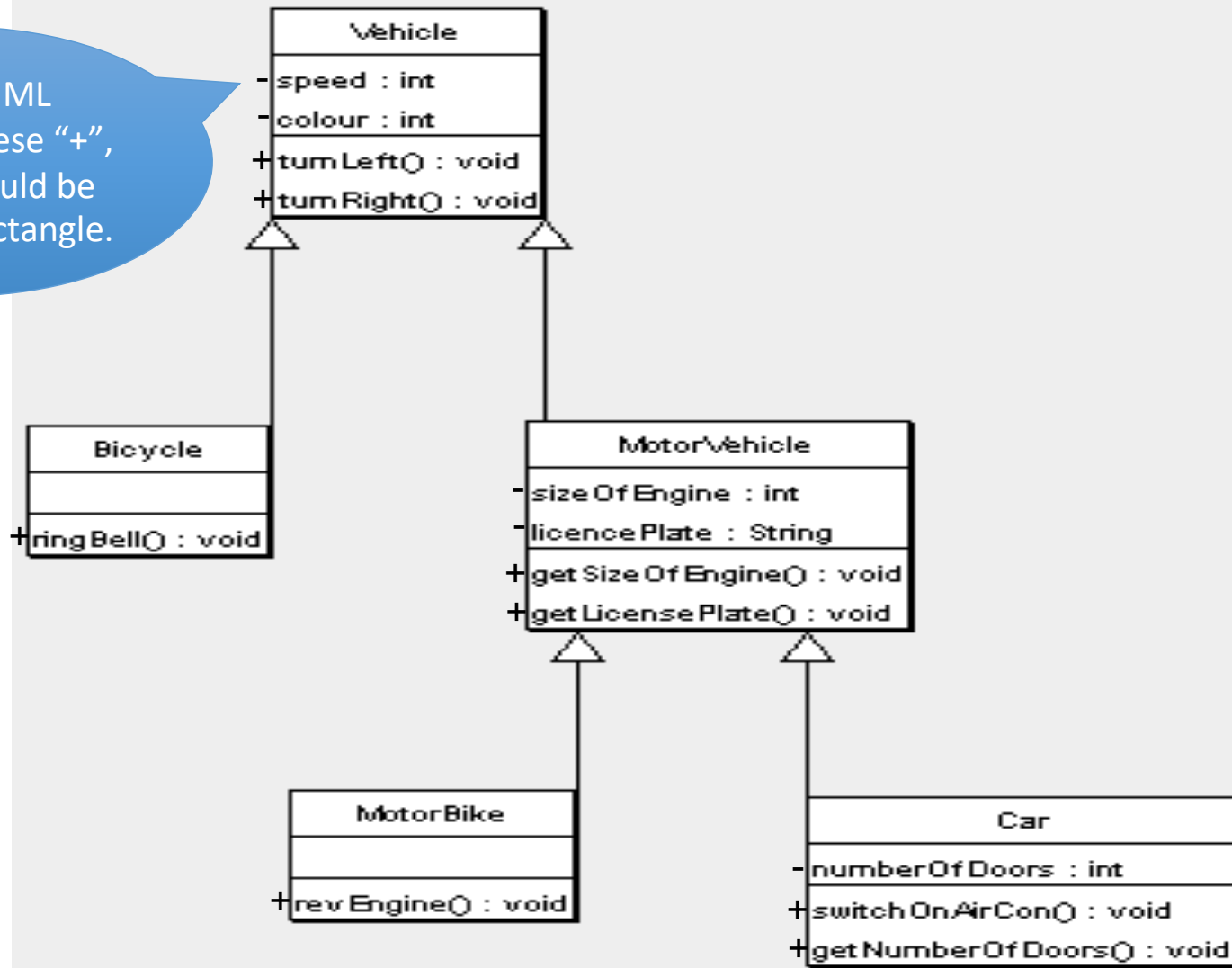


# Interfaces

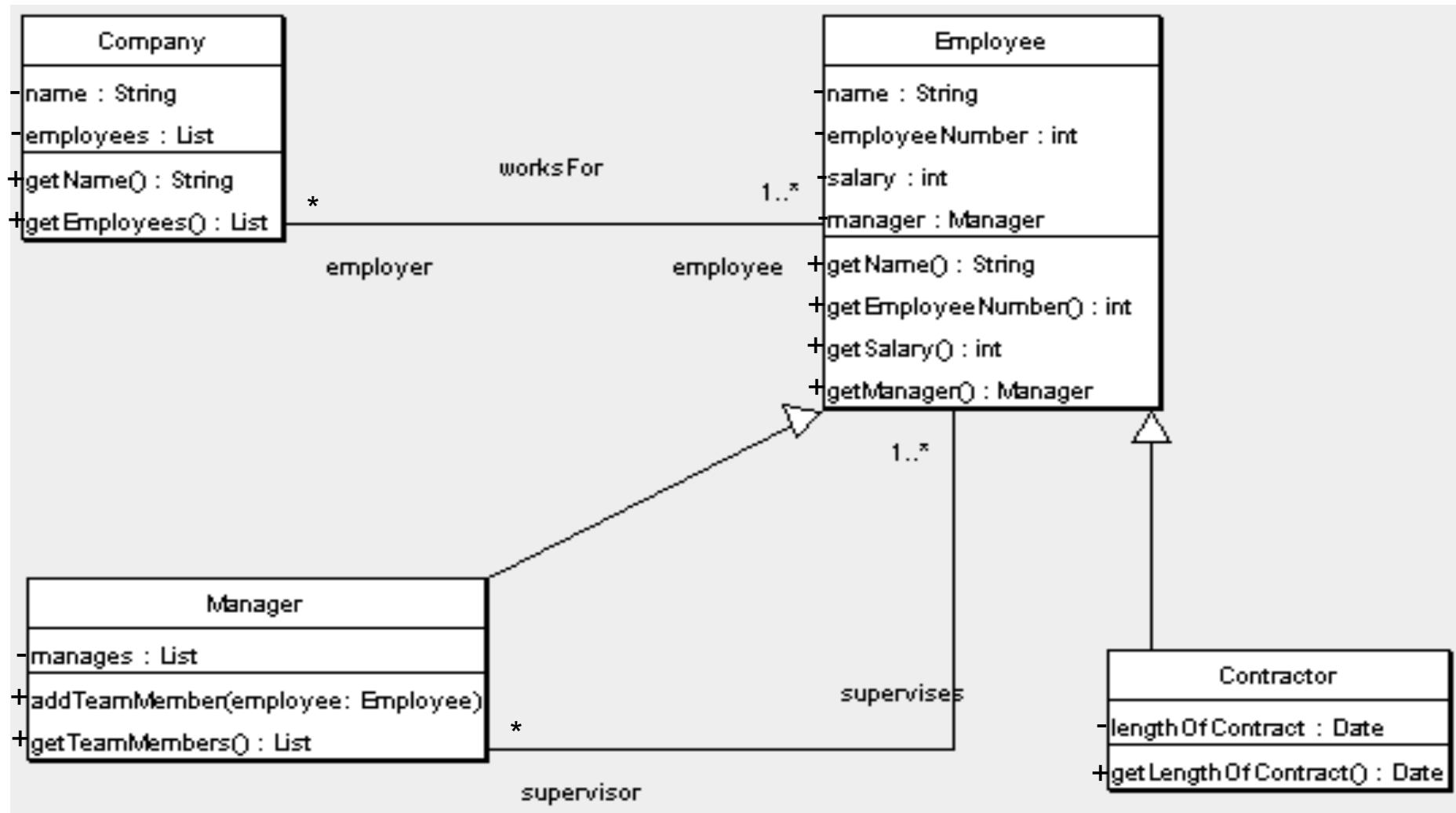


# Sample Class Diagram (1/2)

In your UML diagrams, these "+", "-", etc, should be inside the rectangle.



# Sample Class Diagram (2/2)



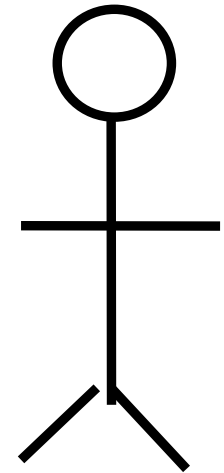
# UML Diagrams: Use Cases

- Means of capturing requirements
  - Used at a very early phase of software development for requirement gathering (analysis phase)
  - Provides a high level overview of the system
  - Class diagrams are created after generating use case diagrams
- Document interactions between user(s) and the system
  - User (actor) is not part of the system itself
  - But an actor can be *another* system
- A scenario based technique in UML
- **Use case diagrams** describe what a system does from the standpoint of an external observer. The emphasis is on *what* a system does rather than *how*



# Actors in Use Case

- What is an Actor?
  - A user or outside system that interacts with the system being designed in order to obtain some value from that interaction
  - It can be a:
    - Human
    - Peripheral device (hardware)
    - External system or subsystem
    - Time or time-based event
  - Labelled using a descriptive noun or phrase
  - Represented by stick figure



# Use Case Analysis (1/4)

- Sample scenario
  - *“A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot”*
- We want to write a use case for this scenario

# Use Case Analysis (2/4)

- Sample scenario

- *“A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot”*

- Who is the actor?

- The actor is a “Patient” here

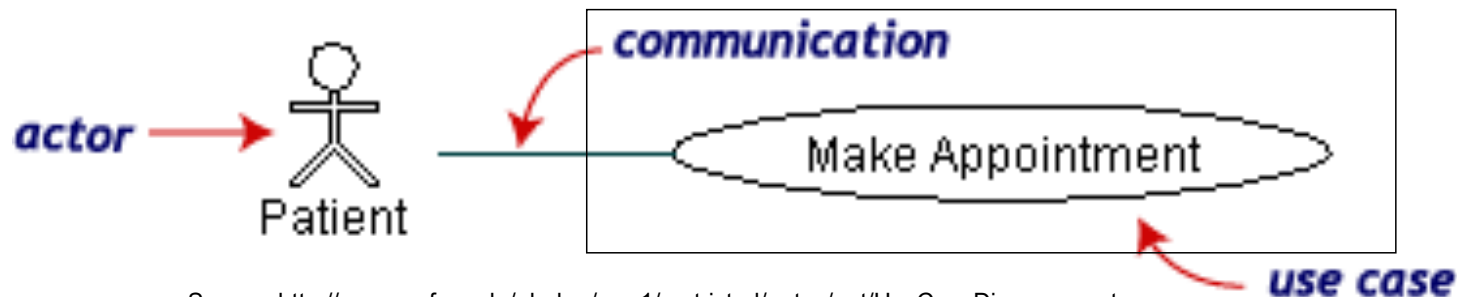


# Use Case Analysis (3/4)

- Sample scenario
  - *“A patient calls the clinic to make an appointment for a yearly checkup. The receptionist finds the nearest empty time slot in the appointment book and schedules the appointment for that time slot”*
- A **use case** is a summary of scenarios for a single task or goal
  - So, what is the use case here?
  - The use case is “Make Appointment”

# Use Case Analysis (4/4)

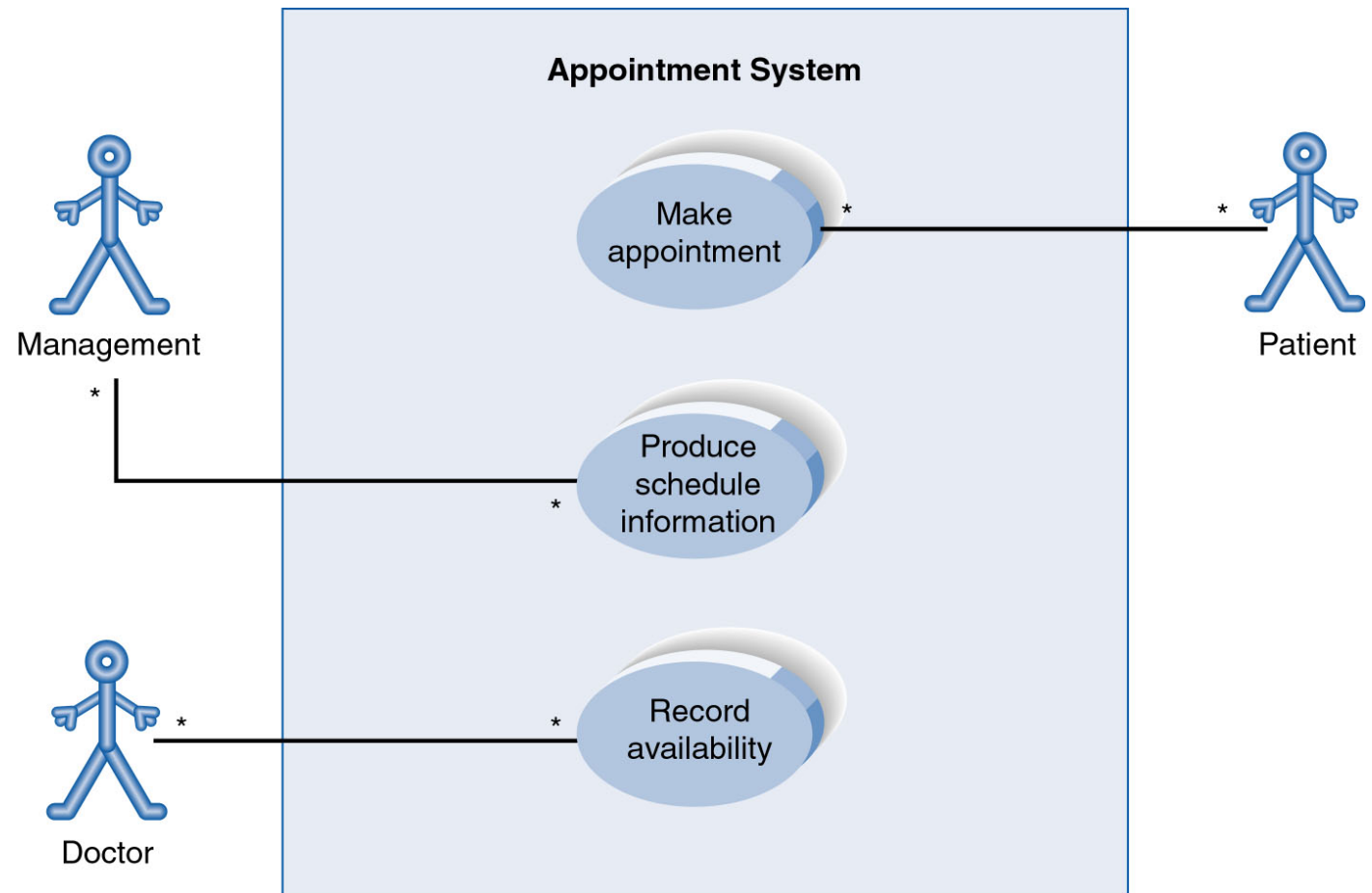
- The picture below is a **Make Appointment** use case for the medical clinic.
- The actor is a **Patient**. The connection between actor and use case is a **communication**
- Actors are stick figures
- Use cases are ovals
  - Labelled using a descriptive verb-noun phrase
- Communications are lines that link actors to use cases
- Boundary rectangle is placed around the perimeter of the system to show how the actors communicate with the system



Source: <http://www.cs.fsu.edu/~baker/swe1/restricted/notes/ppt/UseCaseDiagrams.ppt>

# Use Case Diagram

- A use case diagram is a collection of actors, use cases, and their communications

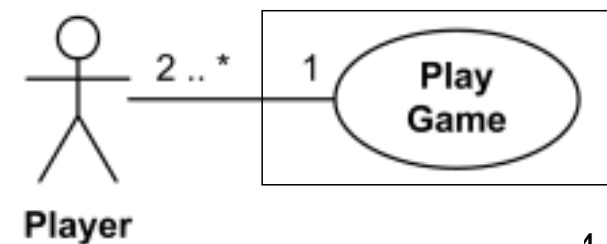
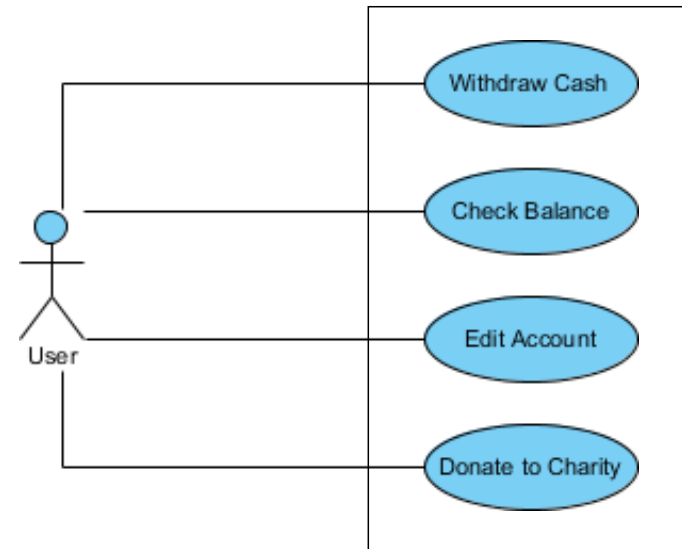


# Relationships for Use Cases

- Association
- Generalization
- Extend
- Include

# Association Relationship

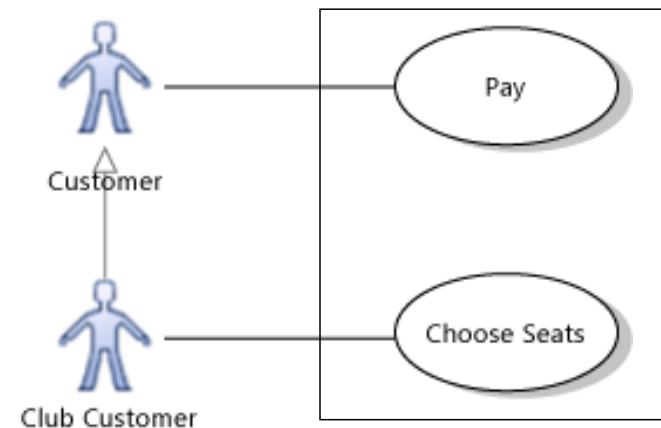
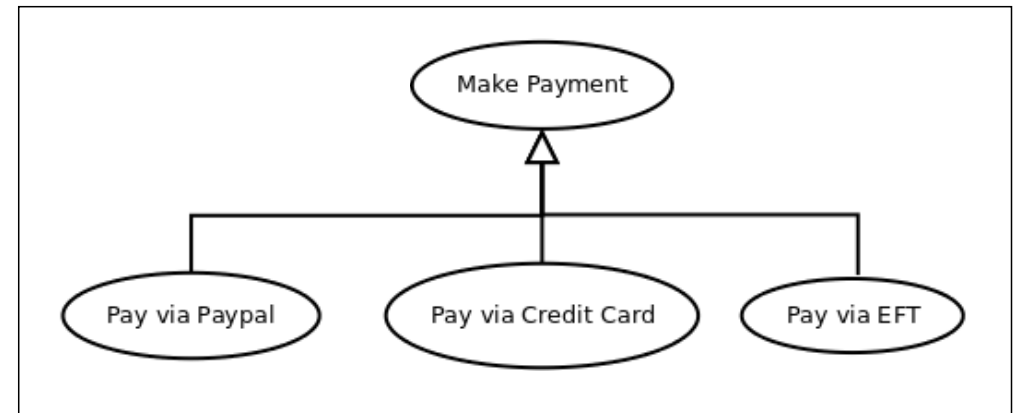
- Exists only between an actor and a use case
  - Indicates that an actor can use certain functionality of the system
- Represented by a solid line without arrowhead
  - Most commonly used representation
  - Uncommon to show one-way association
- The association between an actor and a use case can also show multiplicity at each end





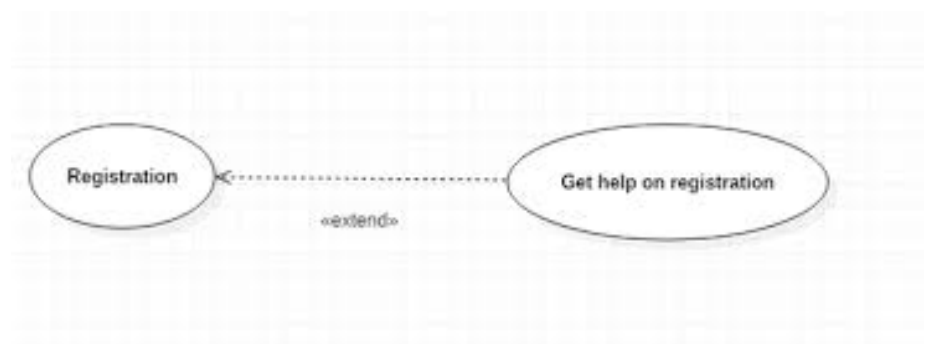
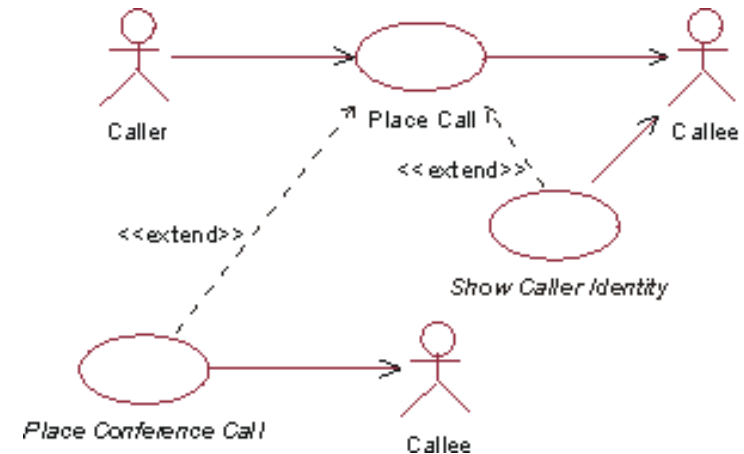
# Generalization Relationship

- Could exist between two actors or between two use cases
  - Indicates parent/child relationship
- Represented by a solid line with a triangular and hollow arrowhead
  - From child to parent



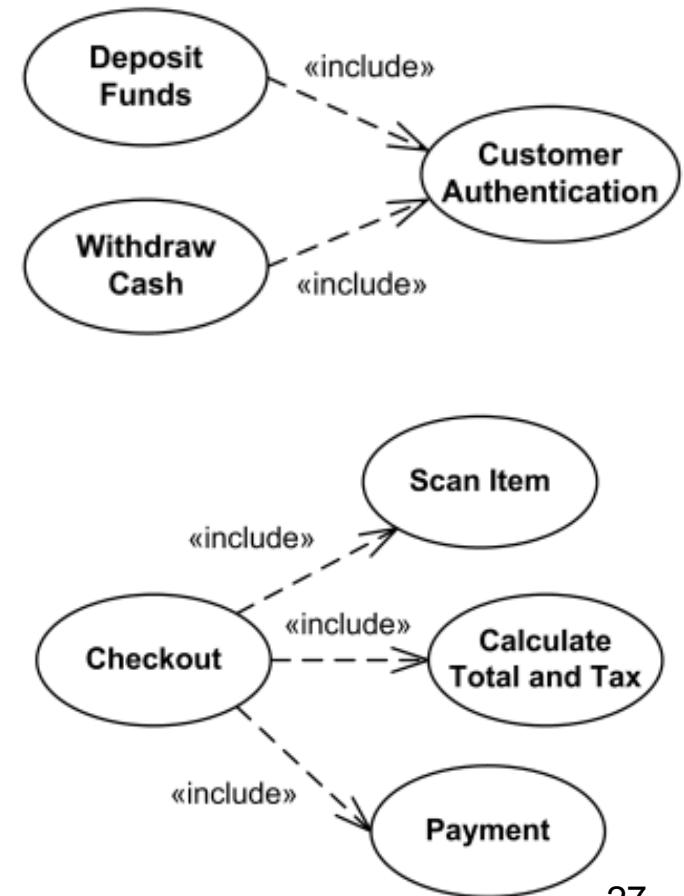
# Extend Relationship “<<extend>”

- Exists only between use cases
  - This relationships represent optional or seldom invoked cases
  - Indicates that although one use case is a variation of another but it is invoked rarely
    - Lot of shared code between these use cases (**not to be confused with inheritance**)
- Represented using a dashed arrow with an arrowhead. The notation “<< extend >>” is also mentioned above the arrow
  - The direction of the arrow is toward the extended use cases

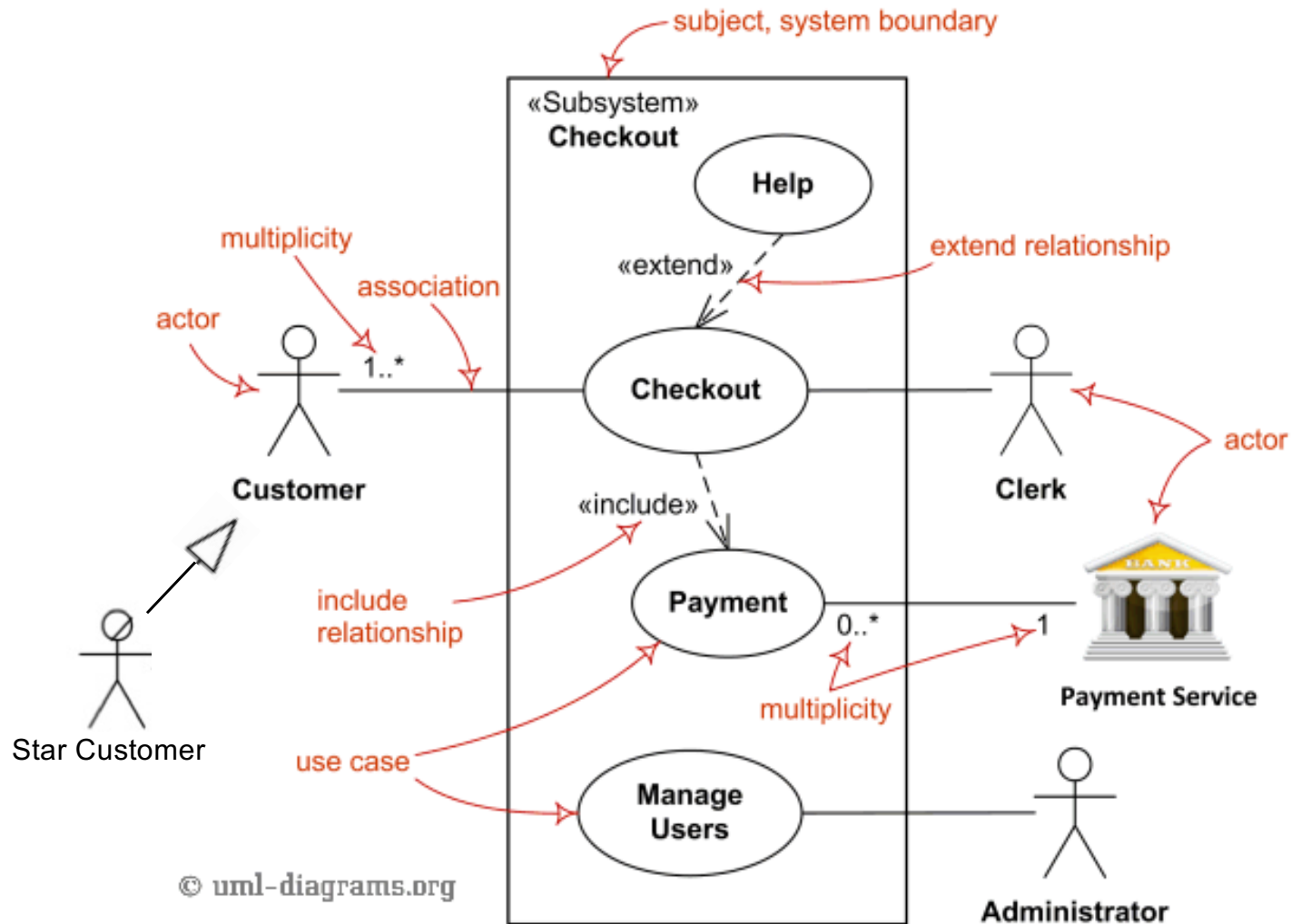


# Include Relationship “<<include>>”

- Exists only between use cases
  - Represents behavior that is factored out of the use case
  - Doesn't mean that the factored out use case is an optional or seldom invoked cases
- Represented using a dashed arrow with an arrowhead. The notation “<< include>>” is also mentioned above the arrow
  - The direction of the arrow is toward the included use case



# Sample Use Case



# Next Lecture

- Event driven programming using JavaFX