Object Oriented Software Engineering (SE 313)
Lab Session # 01

Objective: To Understand Use Case Diagram

Introduction:

Use case analysis is one of the first and primary mean of gathering requirements of the system. Use case diagram captures the functional aspects of the system. it captures the business processes of the system.

A use case is a narrative document that describes the sequence of events of an actor (an external agent) using a system to complete a process." [Jacobson]

"They are stories or cases of using a system. Use case are not exactly requirements or functional specifications, but they illustrate and imply requirements in the stories they tell." [Larman]

UML Use Case Diagrams can be used to describe the functionality of a system in a horizontal way. That is, rather than merely representing the details of individual features of your system, UCDs can be used to show all of its available functionality.

Definition - What Does *Use Case* mean?

A use case is a software and system engineering term that describes how a user uses a system to accomplish a particular goal. A use case acts as a software modeling technique that defines the features to be implemented and
the resolution of any errors that may be encountered.

Explain *Use Case*

Use cases define interactions between external actors and the system to attain particular goals. There are three basic elements that make up a use case:

- **Actors**: Actors are the type of users that interact with the system.
- **System**: Use cases capture functional requirements that specify the intended behavior of the system.
- **Goals**: Use cases are typically initiated by a user to fulfill goals describing the activities and variants involved in attaining the goal.

Use cases are modeled using unified modeling language and are represented by ovals containing the names of the use case. Actors are represented using lines with the name of the actor written below the line. To represent an actor's participation in a system, a line is drawn between the actor and the use case. Boxes around the use case represent the system boundary.

Characteristics associated with use cases are:

Organizing functional requirements

- Modeling the goals of system user interactions
- Recording scenarios from trigger events to ultimate goals
- Describing the basic course of actions and exceptional flow of events
- Permitting a user to access the functionality of another vent The steps in designing use cases are:
- Identify the users of the system
- For each category of users, create a user profile. This includes all roles played by the users relevant to the system.
- Identify significant goals associated with each role to support the system. The system's value proposition identifies the significant role.
- Create use cases for every goal associated with a use case template and maintain the same abstraction level throughout the use case. Higher level use case steps are treated as goals for the lowerlevel.
- Structure the use cases
- Review and validate the users

Where to Use:

UCDs have only 4 major elements: The **actors** that the system you are describing interacts with, the **system** itself, the **use cases, or services**, that the system knows how to perform, and the **lines** that represent relationships between these elements.

Example: A UCD is well suited to the task of describing all of the things that can be done with a database system, by all of the people who might use it (administrators, developers, data entry personnel.)

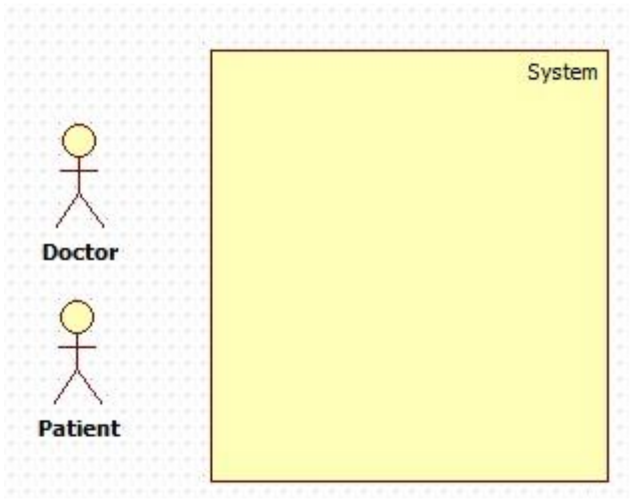
You should NOT use UCDs to represent exception behavior (when errors happen) or to try to illustrate the sequence of steps that must be performed in order to complete a task. Use Sequence diagrams to show these design features.

Example: A UCD would be poorly suited to describing the TCP/IP network protocol, because there are many exception cases, branching behaviors, and conditional functionality (what happens when a packet is lost or late, what about when the connection dies?)

Elements of Use Case Diagram:

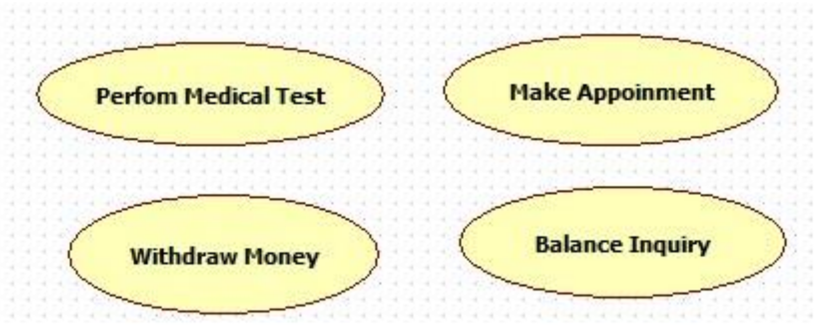
Actors:

An actor is a role that a user plays with respect to the system. Actors carry out use cases. A single actor may perform many use cases; on the other hand, a use case may have several actors performing it. An actor is shown as stick figure in use case diagram depicted "outside" the system boundary as shown in figure



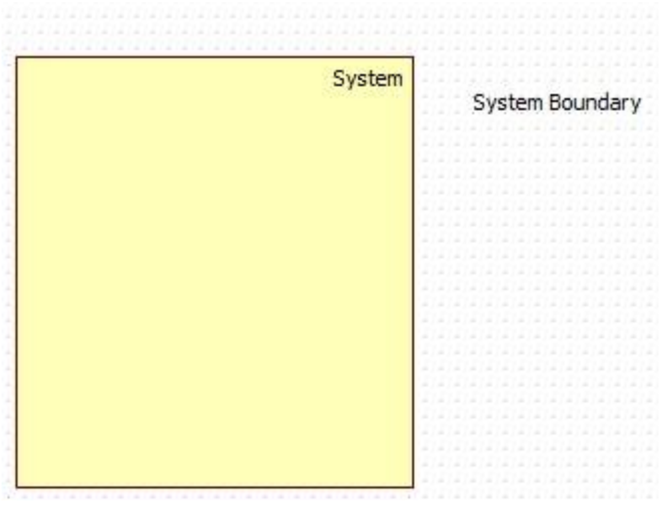
Use Cases:

Use cases in use case diagram are shown as an eclipse. It defines the distinct functionality of the system. To choose a business process as a likely candidate for modeling as use case, ensure that the business process is discrete in nature. As the first step in identifying use case, list the discrete business functions in requirements document. Each of this business function can be classified as a potential use case. A use case can be connected to many actors at a time.

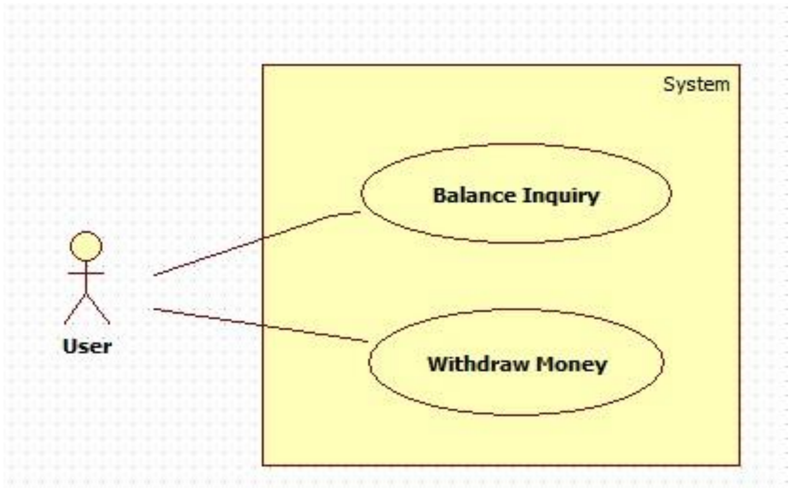


System Boundary:

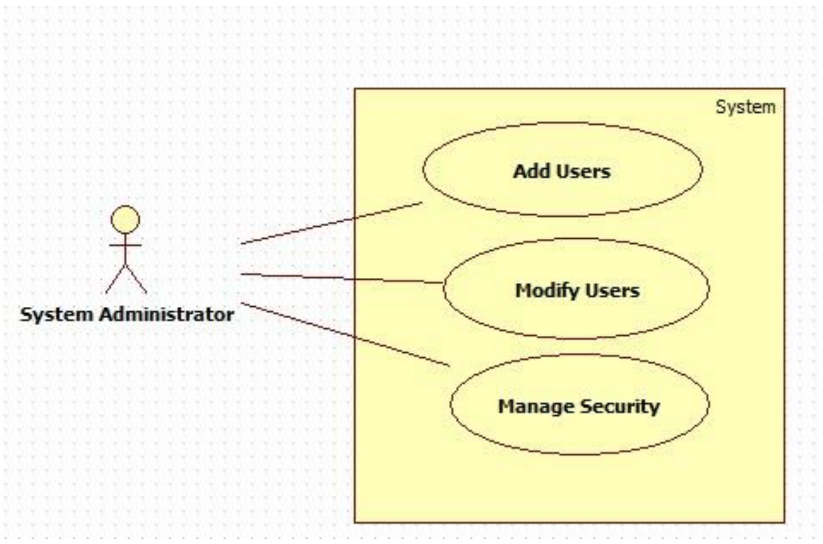
System boundary defines the scope of what a system will be. The system boundary in use case diagram defines the limits of a system. System boundary is shown as a rectangle spanning all the use cases in the system.



Example 01

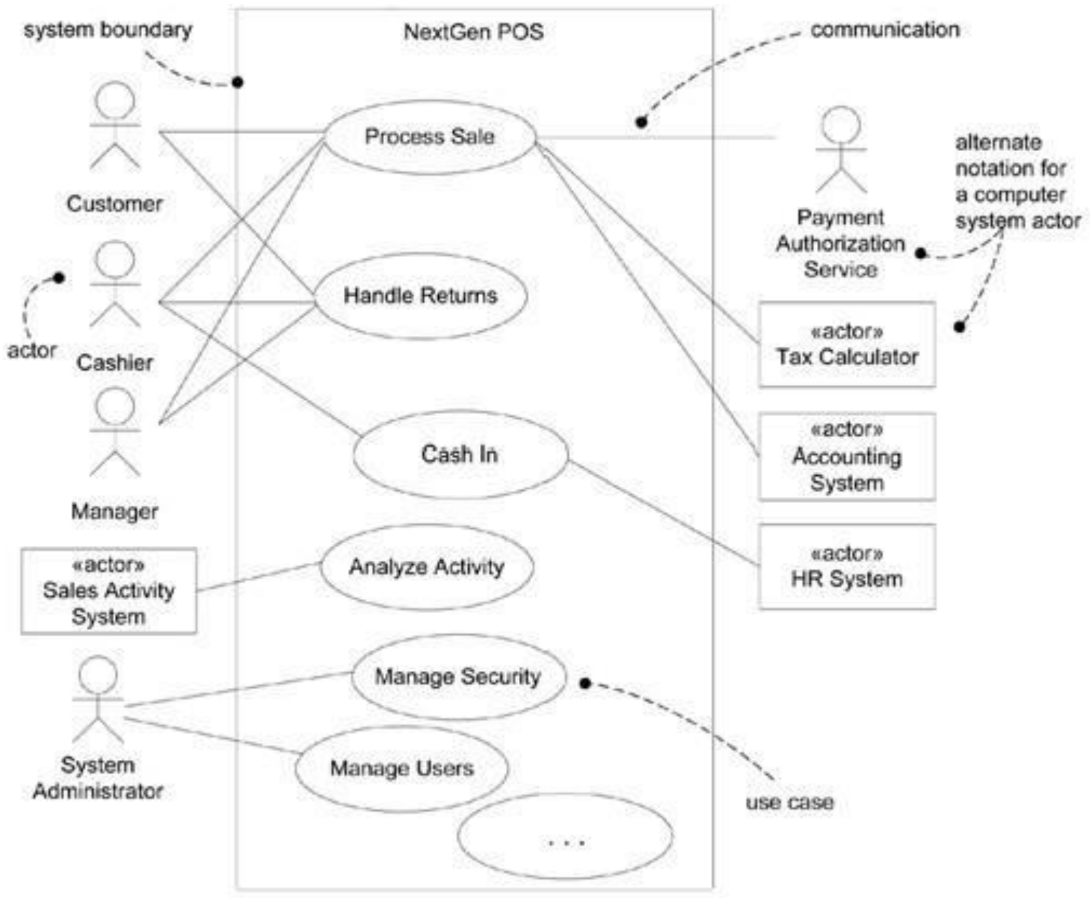


Example 02



Example 03

Partial Use Case Context Diagram



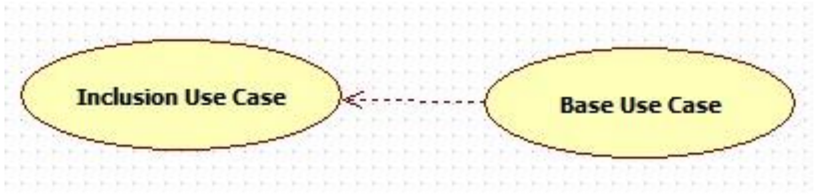
Relationships in Use Case Diagram:

A relationship between two use cases is basically a dependency between two use cases. Use cases share different kind of relationship

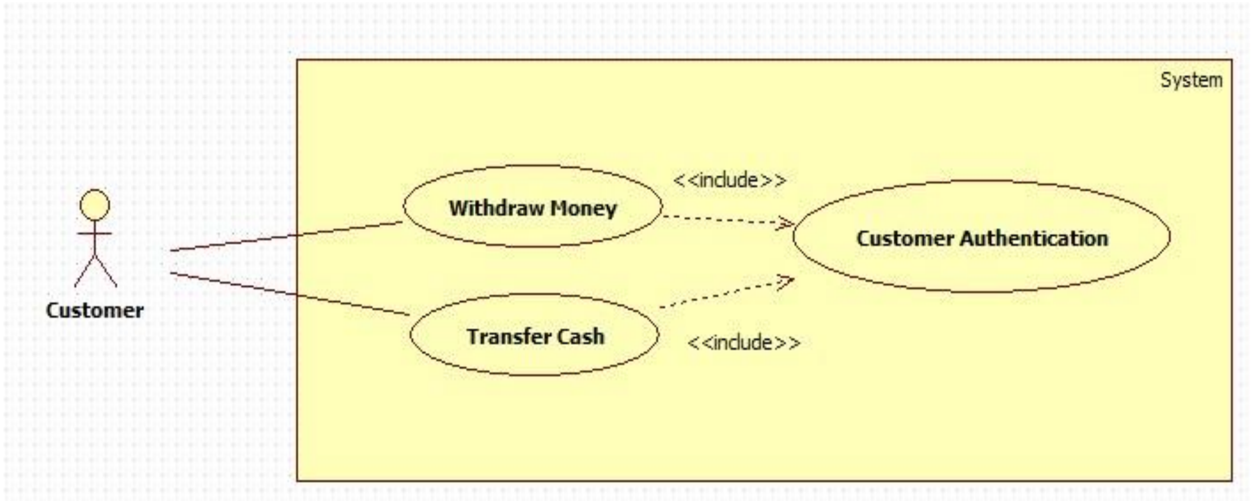
1) Include:

An include relationship is a relationship in which one use case (the base use case) includes the functionality of another use case (the inclusion use case).The include relationship supports the reuse of functionality in a use case.

As the following figure illustrates, an inclusive relationship is shown as a dashed line with an open arrow pointing from the base use case/parent use case to the inclusive use case/child use case. The keyword include is attached to the connector.

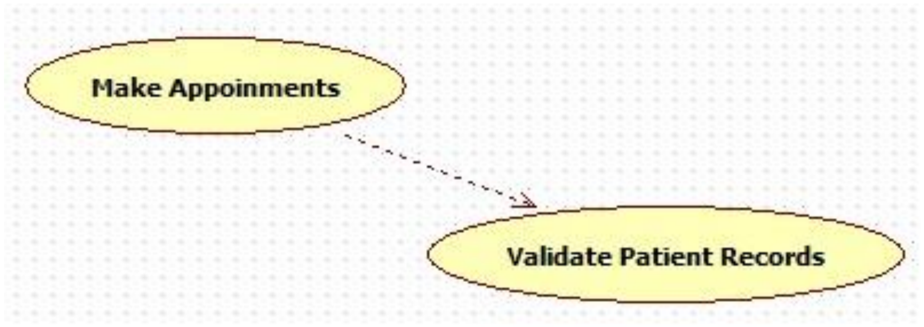


Example 01



Large and complex Checkout use case has several use cases extracted, each smaller use case describing some logical unit of behavior. Note, that including Checkout use case becomes incomplete by itself and requires included use cases to be complete.

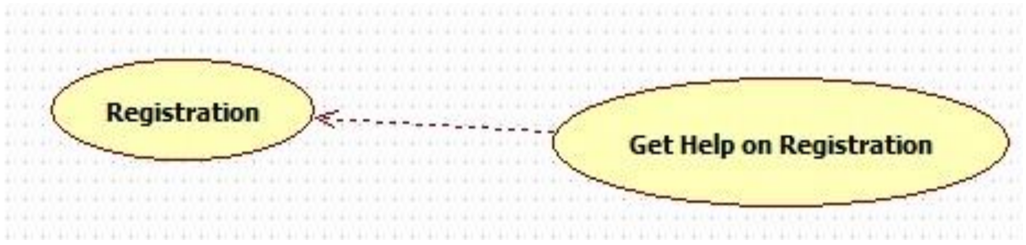
Example 02



For example, in Figure, you can see that the functionality defined by the “Validate patient records" use case is contained within the "Make appointment" use case. Hence, whenever the "Make appointment" use case executes, the business steps defined in the "Validate patient records" use case are also executed.

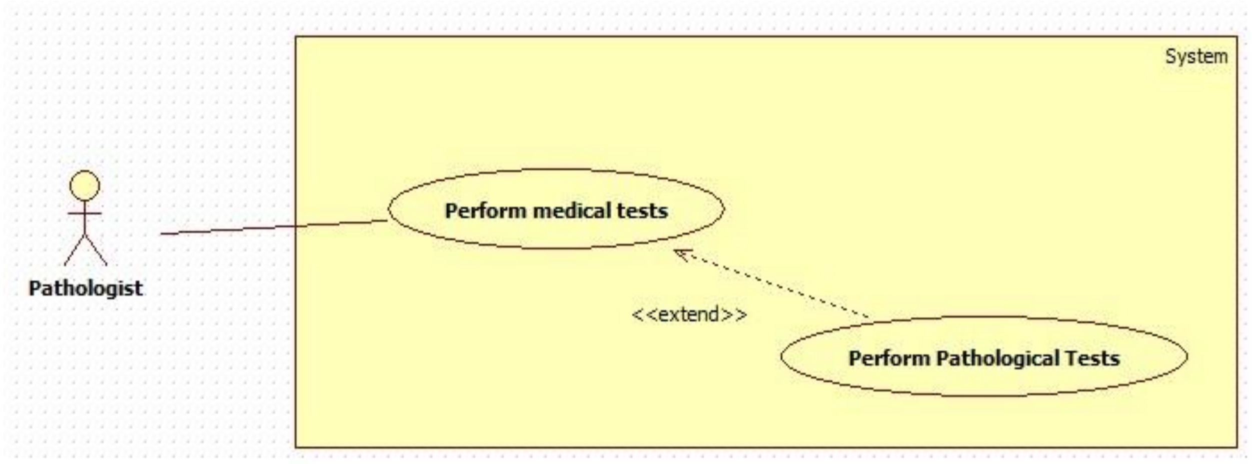
2)Extends

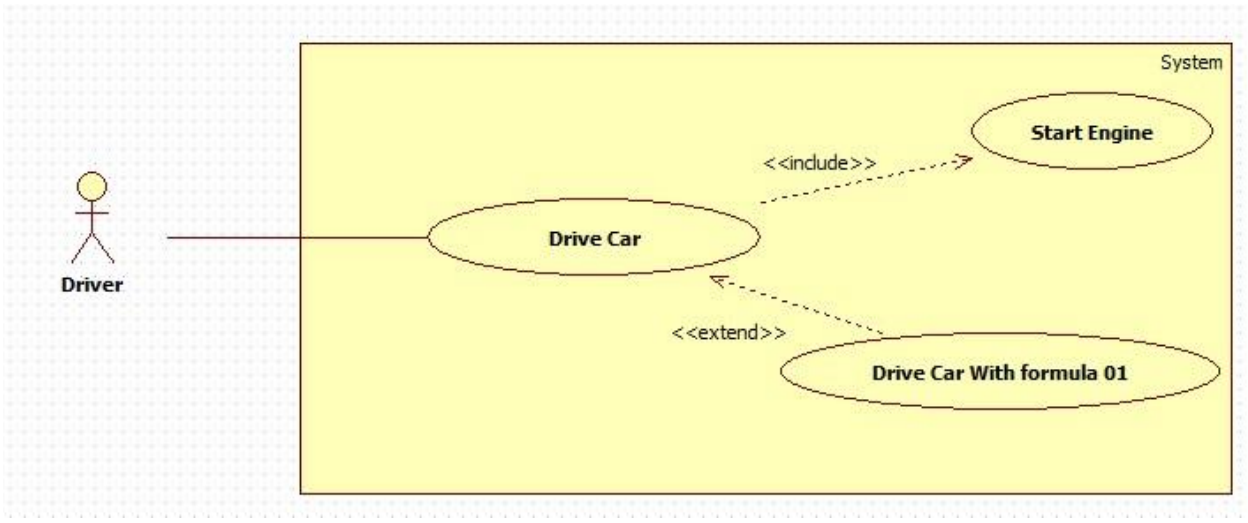
In an extend relationship between two use cases, the child use case adds to the existing functionality and characteristics of the parent use case. An extend relationship is depicted with a directed arrow having a dotted shaft, similar to the include relationship. The tip of the arrowhead points to the parent use case and the child use case is connected at the base of the arrow The keyword extends attached to the connector.



***Registration** use case is complete and meaningful on its own.*
*It could be extended with optional **Get Help On Registration** use case.*

Extended use case is meaningful on its own, it is independent of the extending use case. Extending use case typically defines optional behavior that is not necessarily meaningful by itself. The extend relationship is owned by the extending use case. The same extending use case can extend more than one use case, and extending use case may itself be extended.


System
Pathologist
Perform medical tests
<<extend>>
Perform Pathological Tests


System
Start Engine
<<include>>
Driver
Drive Car
<<extend>>
Drive Car With formula 01

Generalization:

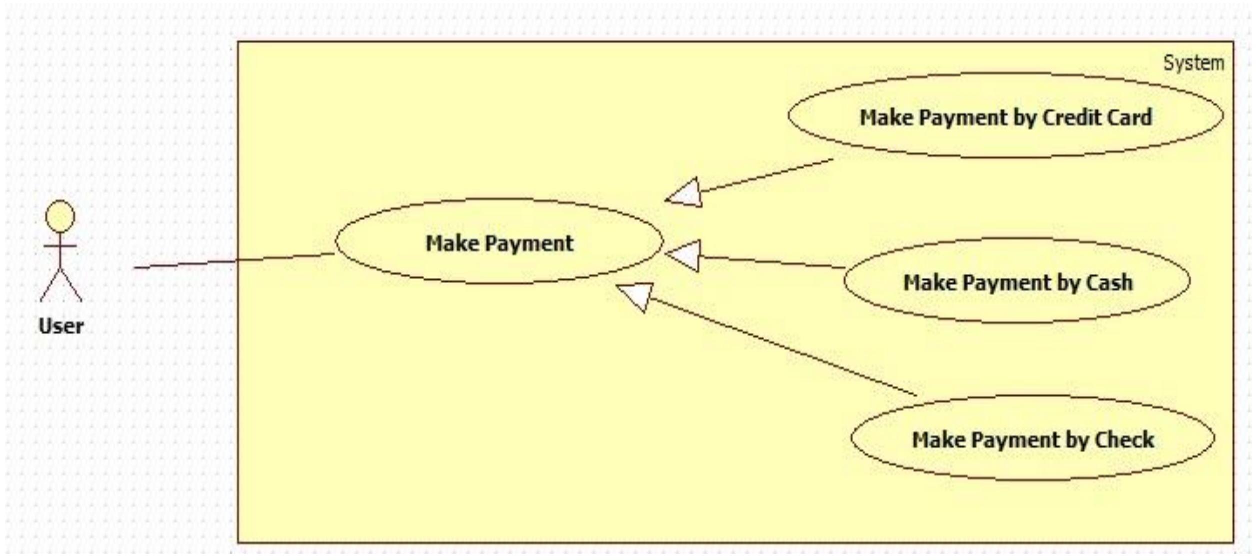
A generalization relationship is a relationship in which one model element (the child) is based on another model element (the parent). The model elements in a generalization relationship must be the same type. For example, a generalization relationship can be used between actors or between use cases; however, it cannot be used between an actor and a use case.

You can add generalization relationships to capture attributes, operations, and relationships in a parent model element and then reuse them in one or more child model elements. Because the child model elements in generalizations inherit the

attributes, operations, and relationships of the parent, you must only define for the child the attributes, operations, or relationships that are distinct from the parent.

Generalization is shown as a directed arrow with a triangle arrowhead. The child use case is connected at the base of the arrow. The tip of the arrow is connected to the parent use case.

Example:


System
Make Payment by Credit Card
Make Payment
Make Payment by Cash
Make Payment by Check
User