Viva Helping Material

FOR

FINAL PROJECT

BY

VU786.COM

Index

Use-case diagrams (UCD)

UCD is created during system analysis phase of Problem Solving Methodology to plan the system's functional requirements: what it should be capable of doing.

Use case diagrams describe what a system does from the standpoint of an external observer.

The main purpose of the use-case diagram is to help development teams visualize the functional requirements of a system.

A use case (the oval shape shown below) illustrates a unit of functionality provided by the system.

Use case diagrams are helpful in:

- Determining features (requirements). New use cases often generate new requirements as the system is analysed and the design takes shape.
- Communicating with clients. Their simplicity makes use case diagrams a good way for developers to communicate with clients.

The main purpose of the UCD is to help development teams visualize the functional requirements of a system, including the relationship of actors to essential processes, as well as the relationships among different use cases.

ACTORS

An actor is a person, organisation, or external system that plays a role in one or more interactions with your system.

An actor is who or what initiates or participates in the events involved in that task.

An actor is always outside the system being modelled. Anything that is part of the system we are defining is not an actor. So don't put the 'database that stores customer information' as an actor on the UCD. It is part of the system and so cannot be an actor.

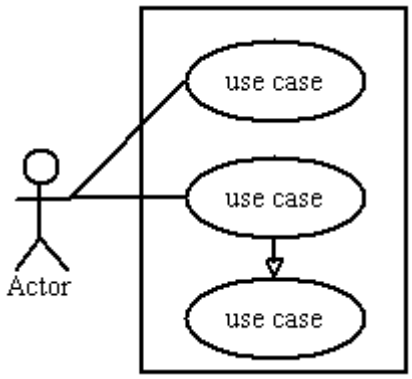
USE CASES

To show a use case on a UCD, draw an oval in the middle of the diagram and put the name of the use case in or below the oval.

Association lines can have an optional arrowhead on one end of the line. The arrowhead is often used to indicate which actor invoked (initiated) the relationship. That actor is called the primary actor within the use case.

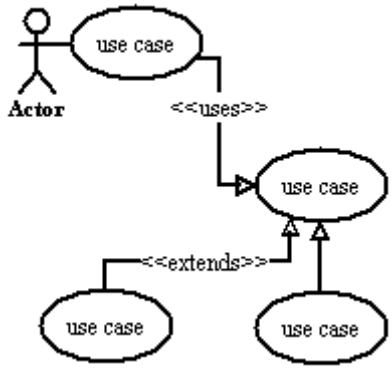
System

Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.



Relationships

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.

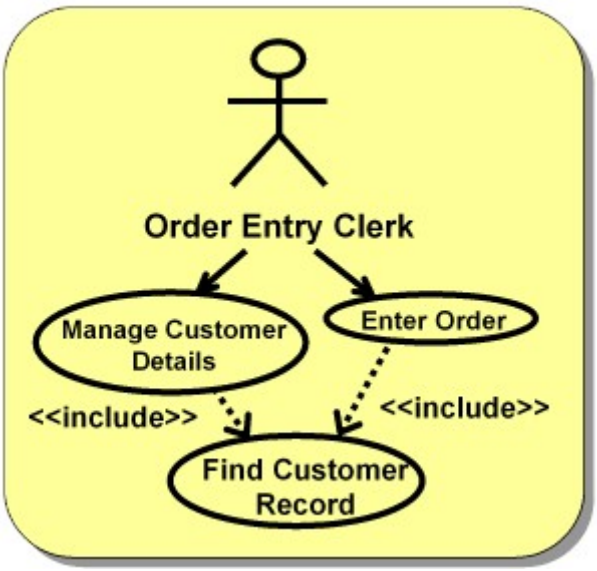


Includes and Extends

<<includes>>

denotes the inclusion of behaviour described by another use case. Think of it like a subprogram. They are always called, unlike extends which are conditionally called. Imagine it as bolting on another set of actions that have already been defined somewhere else – it saves repetition and wasted space.

In the UCD below, both the Manage Customer Details and Enter Order use cases include the Find Customer Record use case.



<<extends>>

Like a conditional <<include>>.A use case that extends another use case (the ‘base’ use case) when circumstances require it.

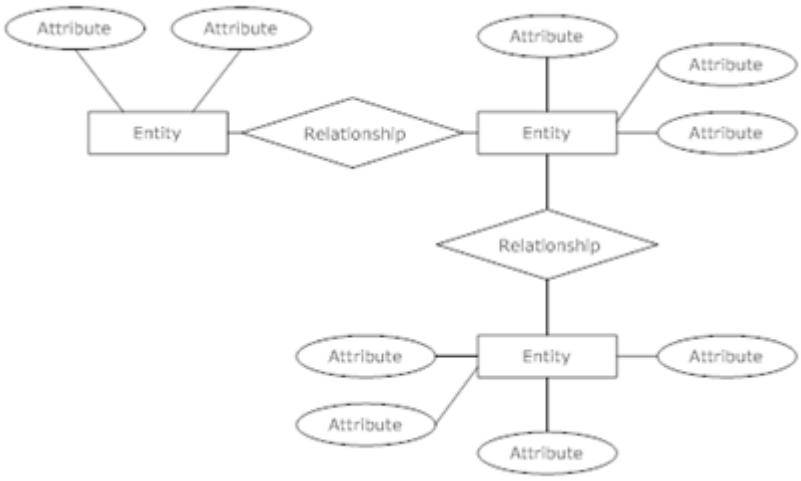
When the extending use case’s activities are finished, the base use case’s activities continue.

The extending use case is really an alternate course of the base use case. It can be used to describe different actions to be taken dependent on current condition. In other words, it can indicate conditional branches in activities.

Entity Relationship Diagram

What are Entity Relationship Diagrams?

Entity Relationship Diagrams (ERDs) illustrate the logical structure of databases.



Entity

An entity is an object or concept about which you want to store information.

Weak Entity

A weak entity is an entity that must defined by a foreign key relationship with another entity as it cannot be uniquely identified by its own attributes alone.

Key attribute

A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute.

Multivalued attribute

A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values.

Derived attribute

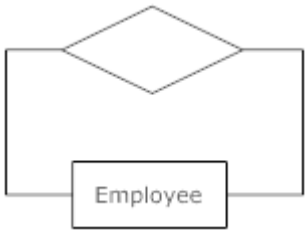
A derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary.

Relationships

Relationships illustrate how two entities share information in the database structure.

Recursive relationship

In some cases, entities can be self-linked. For example, employees can supervise other employees.



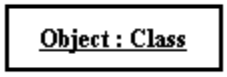
Sequence Diagram

What is a UML Sequence Diagram?

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time.

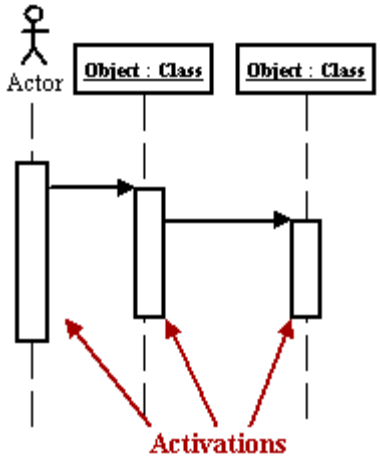
Class roles

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.



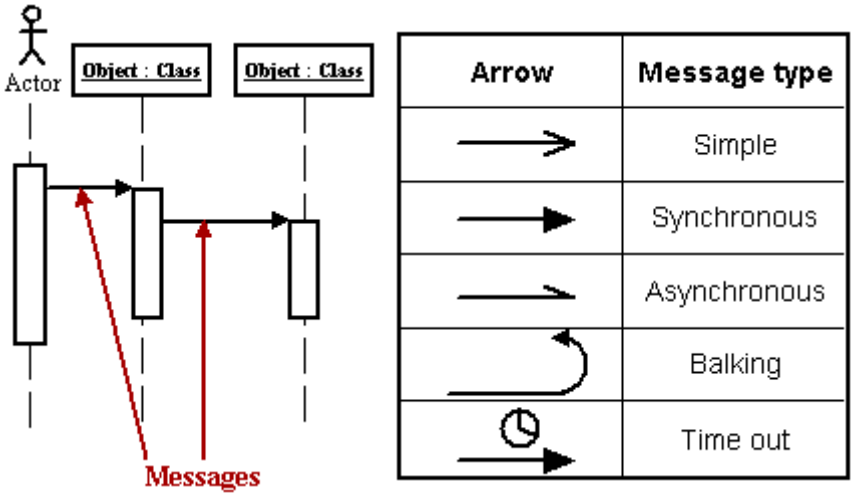
Activation

Activation boxes represent the time an object needs to complete a task.



Messages

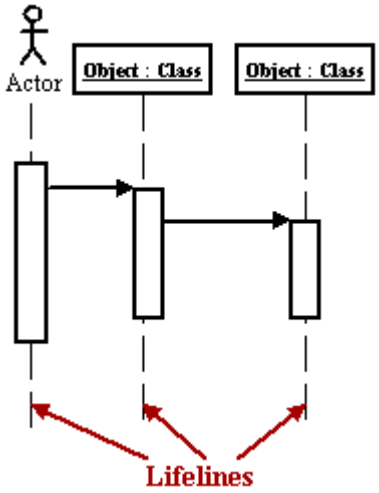
Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks.



Various message types for Sequence and Collaboration diagrams

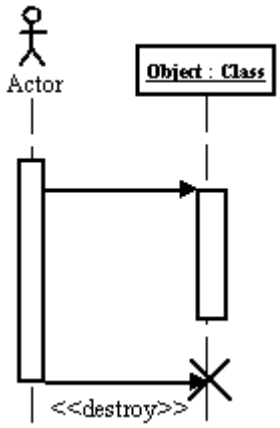
Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



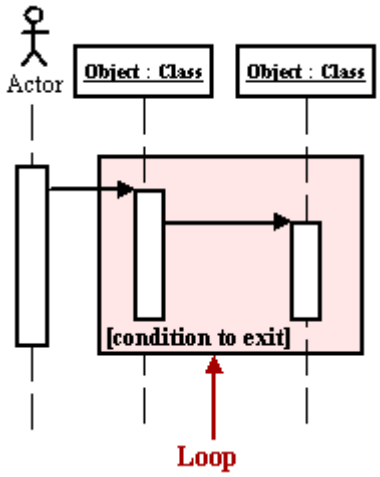
Destroying Objects

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X.



Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].



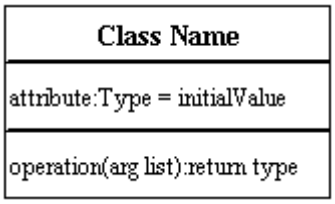
Class Diagram

What is a UML Class Diagram?

Class diagrams are the backbone of almost every object-oriented method including UML. They describe the static structure of a system.

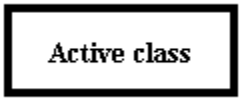
Classes represent an abstraction of entities with common characteristics. Associations represent the relationships between classes.

Illustrate classes with rectangles divided into compartments. Place the name of the class in the first partition (centered, bolded, and capitalized), list the attributes in the second partition, and write operations into the third.



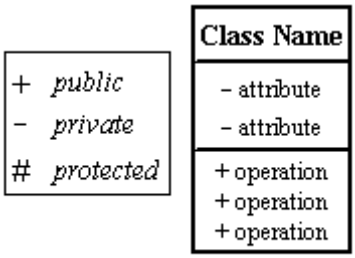
Active Class

Active classes initiate and control the flow of activity, while passive classes store data and serve other classes. Illustrate active classes with a thicker border.



Visibility

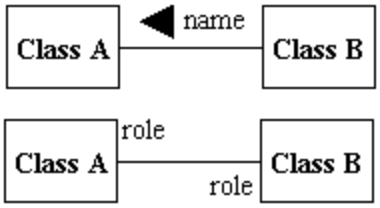
Use visibility markers to signify who can access the information contained within a class. Private visibility hides information from anything outside the class partition. Public visibility allows all other classes to view the marked information. Protected visibility allows child classes to access information they inherited from a parent class.



Associations

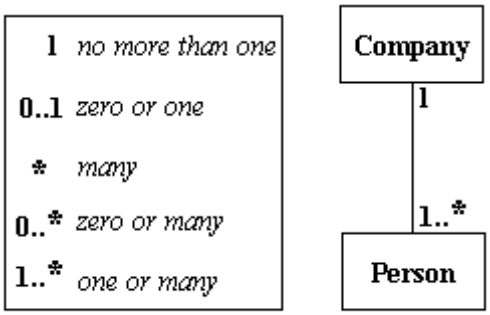
Associations represent static relationships between classes. Place association names above, on, or below the association line. Use a filled arrow to indicate the direction of the relationship. Place roles near the end of an association. Roles represent the way the two classes see each other.

Note: It's uncommon to name both the association and the class roles.



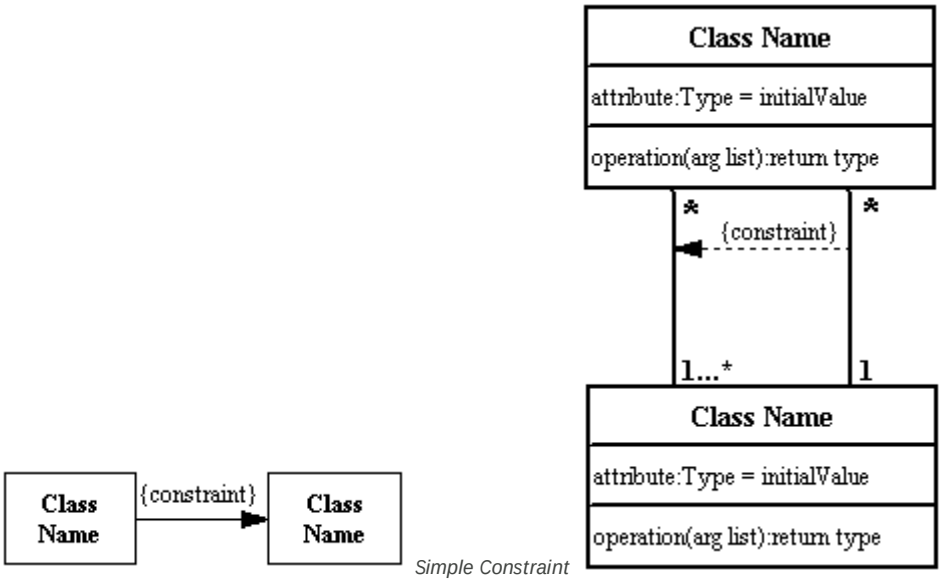
Multiplicity (Cardinality)

Place multiplicity notations near the ends of an association. These symbols indicate the number of instances of one class linked to one instance of the other class. For example, one company will have one or more employees, but each employee works for one company only.



Constraint

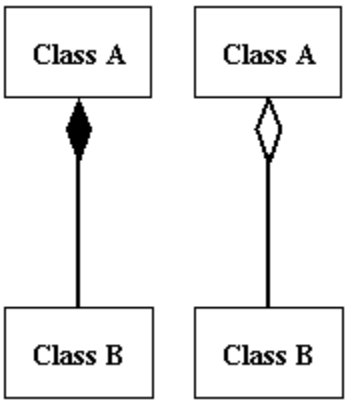
Place constraints inside curly braces {}.



Simple Constraint

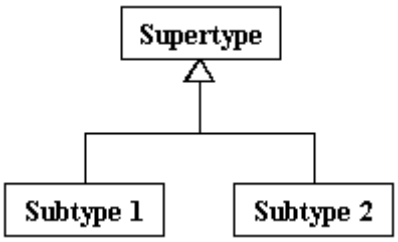
Composition and Aggregation

Composition is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part. Illustrate composition with a filled diamond. Use a hollow diamond to represent a simple aggregation relationship, in which the "whole" class plays a more important role than the "part" class, but the two classes are not dependent on each other. The diamond end in both a composition and aggregation relationship points toward the "whole" class or the aggregate.



Generalization

Generalization is another name for inheritance or an "is a" relationship. It refers to a relationship between two classes where one class is a specialized version of another. For example, Honda is a type of car. So the class Honda would have a generalization relationship with the class car.



In real life coding examples, the difference between inheritance and aggregation can be confusing. If you have an aggregation relationship, the aggregate (the whole) can access only the PUBLIC functions of the part class. On the other hand, inheritance allows the inheriting class to access both the PUBLIC and PROTECTED functions of the superclass.

Activity Diagram

What is a UML Activity Diagram?

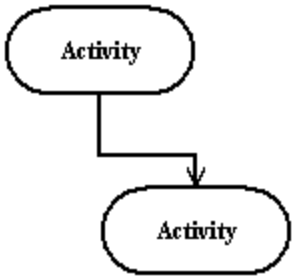
An activity diagram illustrates the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation. Because an activity diagram is a special kind of statechart diagram, it uses some of the same modeling conventions.

Action states

Action states represent the noninterruptible actions of objects. You can draw an action state in SmartDraw using a rectangle with rounded corners.

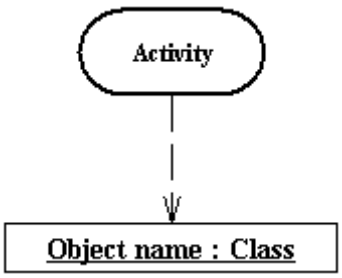
Action Flow

Action flow arrows illustrate the relationships among action states.



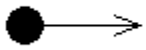
Object Flow

Object flow refers to the creation and modification of objects by activities. An object flow arrow from an action to an object means that the action creates or influences the object. An object flow arrow from an object to an action indicates that the action state uses the object.



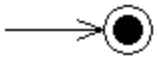
Initial State

A filled circle followed by an arrow represents the initial action state.



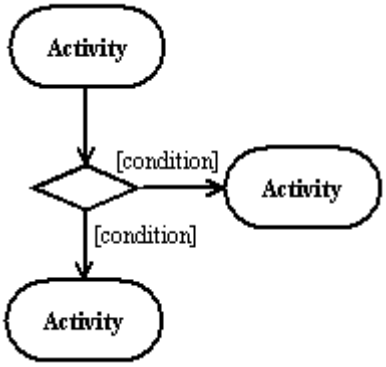
Final State

An arrow pointing to a filled circle nested inside another circle represents the final action state.



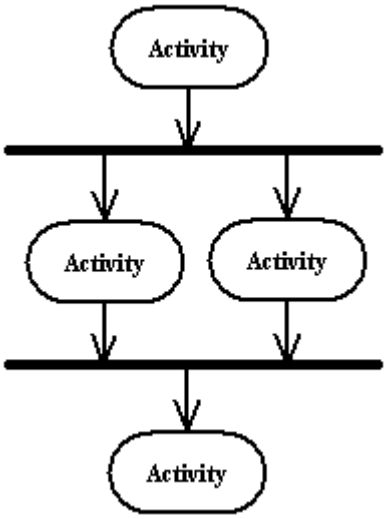
Branching

A diamond represents a decision with alternate paths. The outgoing alternates should be labeled with a condition or guard expression. You can also label one of the paths "else."



Synchronization

A synchronization bar helps illustrate parallel transitions. Synchronization is also called forking and joining.



Swimlanes

Swimlanes group related activities into one column.