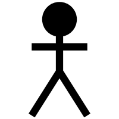
## **Use Cases**

A Use Case is the story of how a system works. It is used in the system analysis to identify, clarify, and organise the system requirements

A use case (or set of use cases) has these characteristics:

* Organises functional requirements
* Models the goals of system/actor (user) interactions
* Records paths (called scenarios) from trigger events to goals
* Describes one main flow of events (also called a basic course of action), and possibly other ones, called exceptional flows of events (also called alternate courses of action)
* Is multi-level, so that one use case can use the functionality of another one.

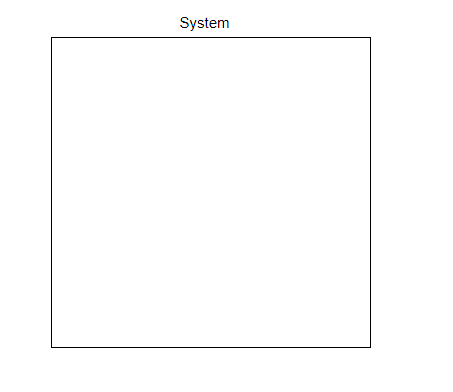
An actor is represented by a stickman in the diagram, and the Actor itself represents anything outside the system that interacts with the system

So, say we have a login system, when the user logins, they interact with the system, so they are interacting with the system therefore they are an actor.

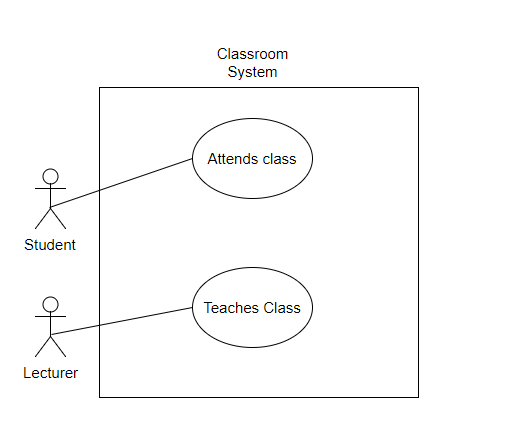
Another would be say we are doing the “Cart” system when you online shop, and the user clicks to pay with Paypal, Paypal is also an actor because it interacts with the system.

There are two parts to a Use Case, there’s the model which is the diagram and the template which is the explanation text.

NB: Make sure to always follow the template form exactly, so that you won’t lose marks!



This is the system boundary, make sure to name it at the top. All the use cases will be inside this, and the actors on the OUTSIDE of the system.



The use cases in the system are surrounded by ellipses and have a noun and a verb.

Below is the template for a use case, make sure to always include the Main flow and give the steps through the system, if there is no alternative flow or exceptional flow make sure to say there isn’t in your template. Also make sure to show where the alternate flow links off, and where it links back in.

# Use case

# Scope

The scope of this use case is

# Description

This use case describes .

# Flow Description

## Precondition

The system is idle.

## Activation

This use case starts when a .

## Main flow

1. .
2. .

## Alternate flow

(A1 )

1. The system

## Exceptional flow

-

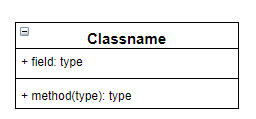
## Termination

The system

## Post condition

The system

## Class Diagrams



In a class diagram you have the Classname, which is the name of it, the attributes go in the field: Type, and the Operations go in the bottom section.

***Objects*** can be any part of any system, could be a machine an organisation a business, a person etc.

There are two types of class diagrams **Conceptual** and **Design**

**Difference between Conceptual and Design:**

A *domain model* is called *conceptual model* in database modeling, while a *design model* is called *logical model*.

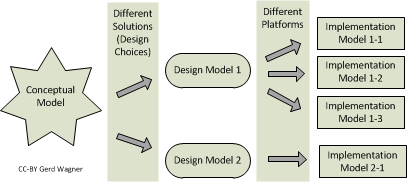
These distinctions are also used in model-driven development, where we have a succession of three types of models:

1. (solution-independent) ***domain models*** resulting from domain/requirements engineering in the system analysis, or inception, phase of a development project
2. (platform-independent) ***design models*** resulting from the system design activities in the elaboration phase
3. (platform-specific) ***implementation models***, which are derived from a design model

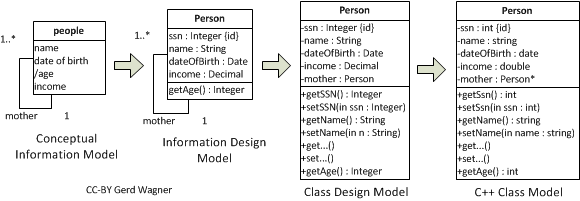
While system modeling includes both information and process modeling, you seem to be concerned with information modeling only. Here, we can use the terms "domain class diagram" and "design class diagram" for the conceptual information model and the information design model made in the form of UML class diagrams.

[The following text/diagrams have been added later, in Sep 2016]

The one-to-many relationships between conceptual models and design models, and between design models and implementation models are illustrated in the following Figure:

[](https://i.stack.imgur.com/VxITP.png)

As an example that illustrates how the derivation chain from concept via design to implementation works, consider the following model of a people/Person concept/class:

[](https://i.stack.imgur.com/LNgb8.png)

Domain models are solution-independent descriptions of a problem domain produced in the analysis phase of a software engineering project. The term "conceptual model" is often used as a synonym of "domain model". A domain model may include both descriptions of the domain’s state structure (in conceptual information models) and descriptions of its processes (in conceptual process models). They are solution-independent, or ‘computation-independent’, in the sense that they are not concerned with making any system design choices or with other computational issues. Rather, they focus on the perspective and language of the subject matter experts for the domain under consideration.

In the design phase, first a platform-independent design model, as a general computational solution to the given software engineering problem, is developed on the basis of the domain model. The same domain model can potentially be used to produce a number of (even radically) different design models representing different design choices. Then, by taking into consideration a number of implementation issues ranging from architectural styles, nonfunctional quality criteria to be maximized (e.g., performance, adaptability) and target technology platforms, one or more platform-specific implementation models are derived from the design model.

See also <http://web-engineering.info/book/WebApp1/ch05s03.html>