### ****Use Case Diagram****

Use case diagrams consist of 4 objects.

* Actor
* Use case
* System
* Package

The objects are further explained below.

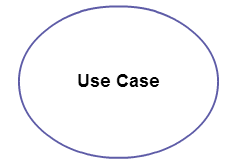
##### Actor

Actor in ause case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below.



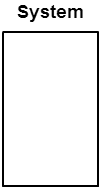
##### Use Case

A use case **represents a function or an action within the system**. Its drawn as an oval and named with the function.



##### System

System is used to **define the scope of the use case** and drawn as a rectangle. This an optional element but useful when your visualizing large systems. For example you can create all the use cases and then use the system object to define the scope covered by your project. Or you can even use it to show the different areas covered in different releases.



##### Package

Package is another optional element that is extremely useful in complex diagrams. Similar to [class diagrams](http://creately.com/diagram-type/class-diagram), packages are **used to group together use cases**. They are drawn like the image shown below.

[](http://static3.creately.com/blog/wp-content/uploads/2014/03/Package1.png)

### Use Case Diagram Guidelines

Although use case diagrams can be used for various purposes there are some common guidelines you need to follow when drawing use cases.

These include naming standards, directions of arrows, placing of use cases, usage of system boxes and also proper usage of relationships.

Relationships in Use Case Diagrams

There are five types of relationships in a use case diagram. They are

* Association between an actor and a use case
* Generalization of an actor
* Extend relationship between two use cases
* Include relationship between two use cases
* Generalization of a use case

### How to Create a Use Case Diagram

Up to now you’ve learned about objects, relationships and guidelines that are critical when drawing use case diagrams. I’ll explain the various processes using a banking systems as an example.

##### Identifying Actors

Actors are external entities that interact with your system. It can be a person, another system or an organization. In a banking system the most obvious actor is the customer. Other actors can be bank employee or cashier depending on the role your trying to show in the use case.

An example of an external organization can be the tax authority or the central bank. Loan processor is a good example of external system associated as an actor.

##### Identifying Use Cases

Now it’s time to identify the use cases. A good way to do this is to identify what the actors needs from the system. In a banking system a customer will need to open accounts, deposit and withdraw funds, request check books and similar functions. So all of these can be considered as use cases.

Top level use cases should always provide a complete functions required by an actor. You can extend or include use cases depending on the complexity of the system.

Once you identify the actors and the top level use case you have basic idea of the system. Now you can fine tune it and add extra layers of detail to it.

##### Look for Common Functionality to use Include

Look for common functionality that can be reused across the system. If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. Then you can connect it via the include relationship to show that its always called when the original use case is executed. ( see the diagram for an example ).

##### Is it Possible to Generalize Actors and Use Cases

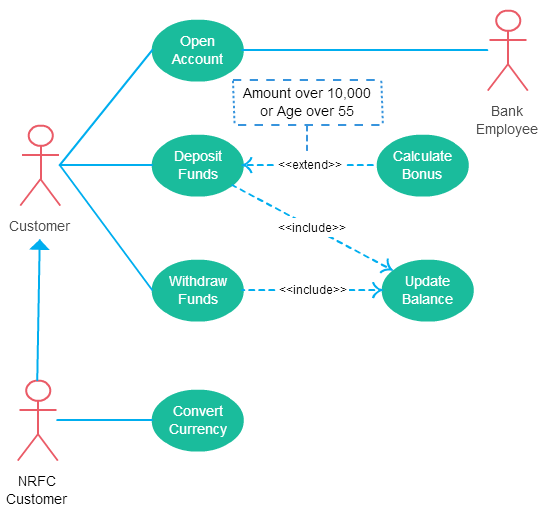
There maybe instances where actors are associated with similar use cases while triggering few use cases unique only to them. In such instances you can generalize the actor to show the inheritance of functions. You can do a similar thing for use case as well.

One of the best examples of this is “Make Payment” use case in a payment system. You can further generalize it to “Pay by Credit Card”, “Pay by Cash”, “Pay by Check” etc. All of them have the attributes and the functionality of a payment with special scenarios unique to them.

##### Optional Functions or Additional Functions

There are some functions that are triggered optionally. In such cases you can use the extend relationship and attach and extension rule to it. In the below banking system example “Calculate Bonus” is optional and only triggers when a certain condition is matched.

Extend doesn’t always mean its optional. Sometimes the use case connected by extend can supplement the base use case. Thing to remember is that the base use case should be able to perform a function on its own even if the extending use case is not called.

[](http://static3.creately.com/blog/wp-content/uploads/2015/02/use-case-diagram-relationships-include.png)

A use case with most of scenarios found in use case diagrams

**Airport Check-In and Security Screening**

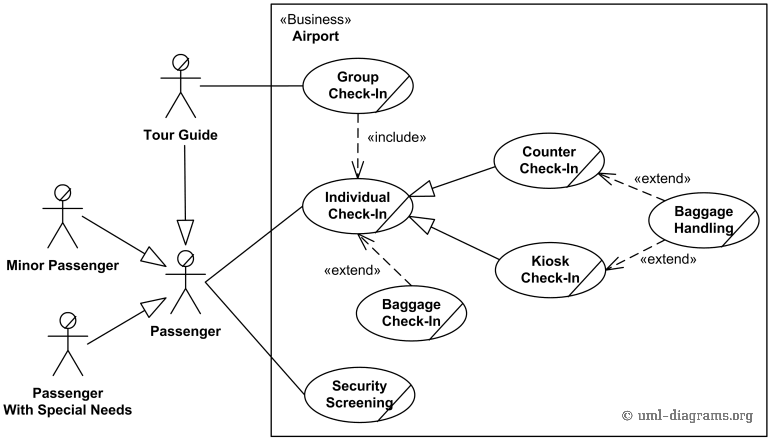
**UML Use Case Diagram Example**

This is an example of a [business use case](http://www.uml-diagrams.org/use-case.html#business-use-case) diagram which is usually created during **Business Modeling** and is rendered here in **Rational Unified Process** (RUP) notation.

[Business actors](http://www.uml-diagrams.org/use-case-actor.html#business-actor) are Passenger, Tour Guide, Minor (Child), Passenger with Special Needs (e.g. with disabilities), all playing **external roles** in relation to airport business.

[Business use cases](http://www.uml-diagrams.org/use-case.html#business-use-case) are Individual Check-In, Group Check-In (for groups of tourists), Security Screening, etc. - representing business functions or processes taking place in airport and serving the needs of passengers.

Business use cases Baggage Check-in and Baggage Handling extend Check-In use cases, because passenger might have no luggage, so baggage check-in and handling are optional.



# Hospital Management

## UML Use Case Diagram Example

**Hospital Management System** is a large system including several subsystems or modules providing variety of functions. UML use case diagram example below shows actor and use cases for a hospital's reception.

**Purpose:** Describe major services (functionality) provided by a hospital's reception.

**Hospital Reception** subsystem or module supports some of the many job duties of hospital receptionist. Receptionist schedules patient's appointments and admission to the hospital, collects information from patient upon patient's arrival and/or by phone. For the patient that will stay in the hospital ("inpatient") she or he should have a bed allotted in a ward. Receptionists might also receive patient's payments, record them in a database and provide receipts, file insurance claims and medical reports.

