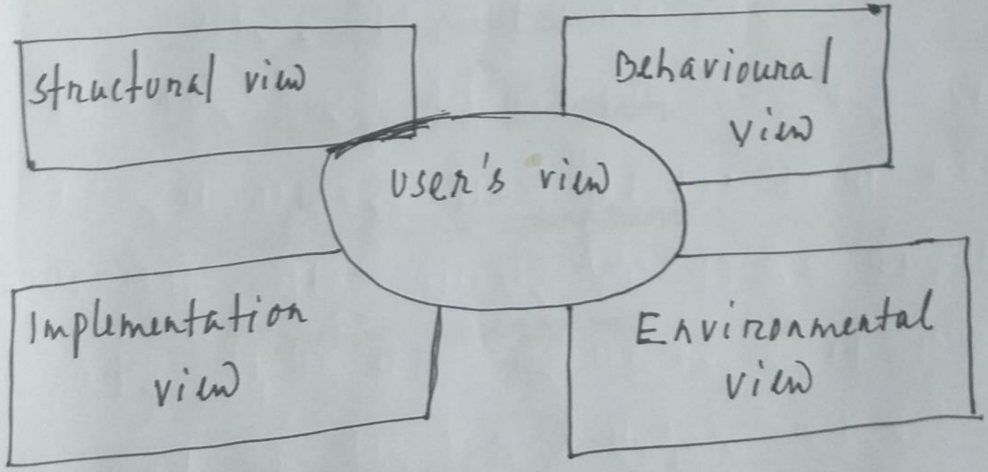
MODULE-2

5 views of UML

1. User's view or use case view
2. Structural view or logical view or Design view
3. Behavioural view or process view
4. Implementation view
5. Deployment view or Environmental view.



UML diagrams correspond to different views

1. User's view
   - ↳ Use case diagram

2. Structural view
   - ↳ Class diagram
   - ↳ Object diagram
   - ~~↳ package diagram~~
   - ~~↳ Composite Structure diagram~~

3. Behavioural view
   - ↳ Sequence diagram
   - ↳ Collaboration diagram
   - ↳ State-Chart diagram
   - ↳ Activity diagram

4. Implementation view
   - ↳ Component diagram

5. Environmental view
   - ↳ Deployment diagram.

Use-Case model

→ Use cases describe how a system interacts with outside users

→ Use cases represents different functionalities that a system provides to its users.

→ It is a useful model for requirement gathering. (i.e. identifying, clarifying and organizing system requirements)

→ Use-case model shows a view of the system from the user's perspective, thus describing what a system does without describing how the system does it i.e. it is free of techical or implementation details.

→ The key elements of a use-case model are

- Actors
- use cases
- and relationships

→ The UML representation of a use-case model is Use-case diagram (i.e. Graphical representation of use-case model)

purpose of use-case diagrams

Use-case diagrams are typically developed at early stages of development for the following purposes

- to specify the context of a system
- to capture the requirement of a system
- to validate a system's architecture
- ~~de~~ to derive implementation and generate test cases

Use-case diagrams are generally developed by analysts together with domain experts.

Actors

→ An actor is a direct external user of a system that communicates with the system but not part of the system.

→ Actors can be persons, devices and other systems.

e.g. Customer and repair technician are different actors of a vending machine.

for a computer database system actor might include user and administrator

→ Basically, actor is an object or set of objects that communicates directly with the system but that is not the part of the system.

UML specifies that an actor in a use-case is a class and the individuals are instances (objects) of that class.

Use Cases

→ use case describes how actors uses a system to accomplish a particular goal.

→ A use case is a coherent piece of functionality that a system can provide by interacting with actors.

e.g. a customer actor buy a beverage from a vending machine. The customer insert money into the machine, makes a selection and ultimately receives the beverages.

Similarly, a repair technician actor can perform scheduled maintenance on a vending machine.

→ A use case involves one or more actors as well as the system itself.

e.g. the use case buy a beverage involves the customer actor and the use case perform scheduled maintance involves repair technician actor.

→ A use case involves sequence of messages among the system and its actors.

for example in buy a beverage use case the sequence of messages are

- insert money (customer insert money)
- Displaying deposited amount (system displays)
- selecting item (customer select the item by pushing respective button)
- Beverage dispensing (machine dispenses the beverage and issue change if necessary)

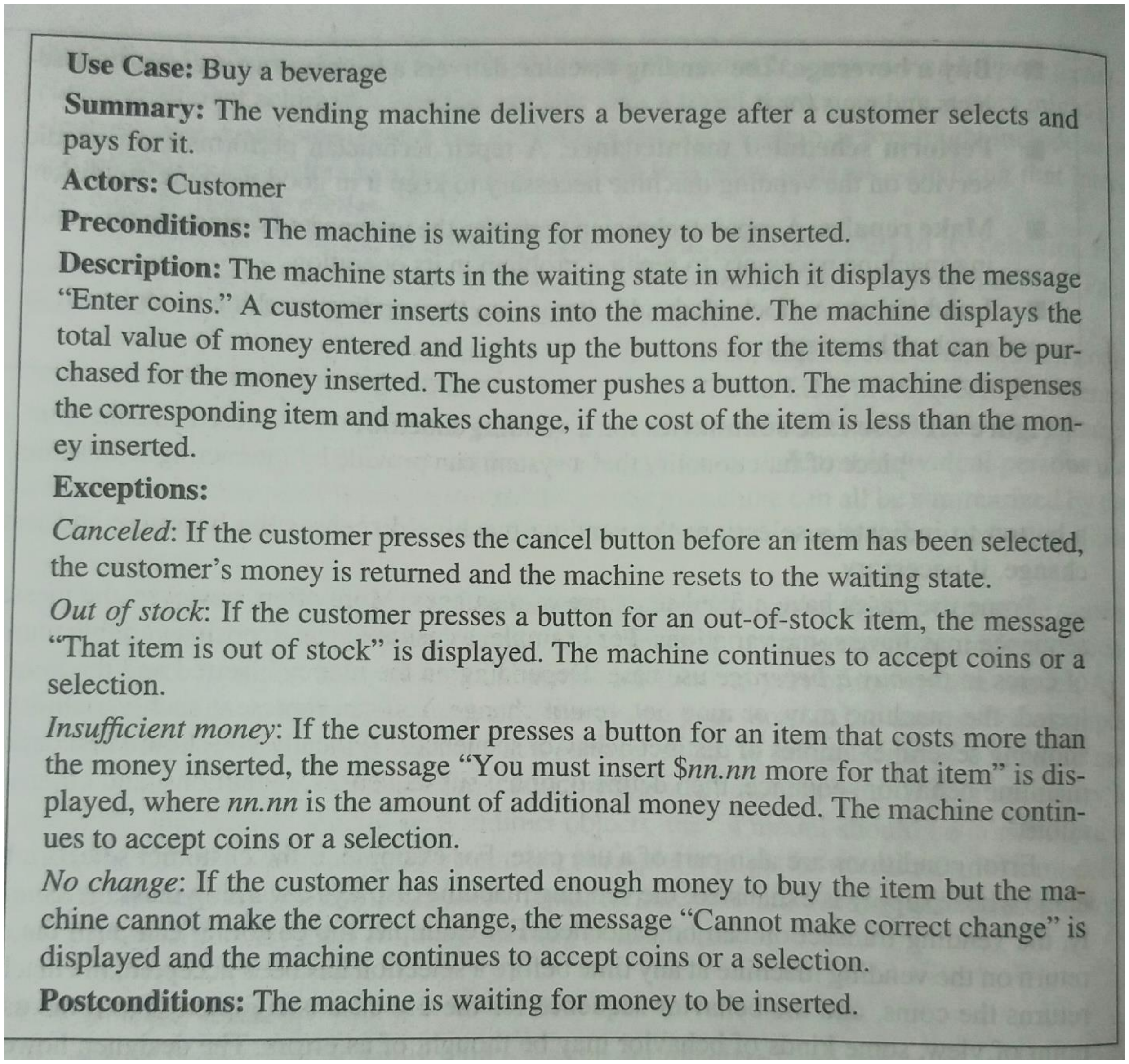
→ Error Conditions are also part of a use case.

e.g. if a customer selects a beverage whose supply is exhausted, the vending machine displays warning message

e.g. Before selection, if coins are pushed to vending machine, vending machine returns the coin.

→ A use case brings together all the behaviour relevant to the system functionality. This includes normal or standard behaviour, variations on normal behaviour, exception conditions, ~~abnormal conditions~~ error conditions, and cancellation of request.

The following is the formal description of **Buy a beverage** use cae.



Use Case: Buy a beverage

Summary: The vending machine delivers a beverage after a customer selects and pays for it.

Actors: Customer

Preconditions: The machine is waiting for money to be inserted.

Description: The machine starts in the waiting state in which it displays the message "Enter coins." A customer inserts coins into the machine. The machine displays the total value of money entered and lights up the buttons for the items that can be purchased for the money inserted. The customer pushes a button. The machine dispenses the corresponding item and makes change, if the cost of the item is less than the money inserted.

Exceptions:

Canceled: If the customer presses the cancel button before an item has been selected, the customer's money is returned and the machine resets to the waiting state.

Out of stock: If the customer presses a button for an out-of-stock item, the message "That item is out of stock" is displayed. The machine continues to accept coins or a selection.

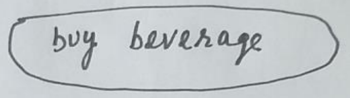
Insufficient money: If the customer presses a button for an item that costs more than the money inserted, the message "You must insert $*nn.nn* more for that item" is displayed, where *nn.nn* is the amount of additional money needed. The machine continues to accept coins or a selection.

No change: If the customer has inserted enough money to buy the item but the machine cannot make the correct change, the message "Cannot make correct change" is displayed and the machine continues to accept coins or a selection.

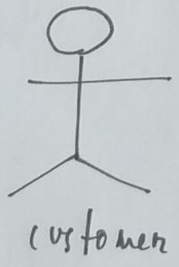
Postconditions: The machine is waiting for money to be inserted.

Use - case diagrams

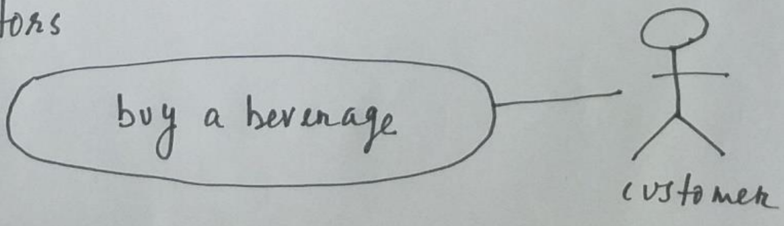
- A rectangle contains the use cases for a system
- The name of the system is written near a side of the rectangle
- A use case is represented by ellipse and name of the use case is specified within the ellipse



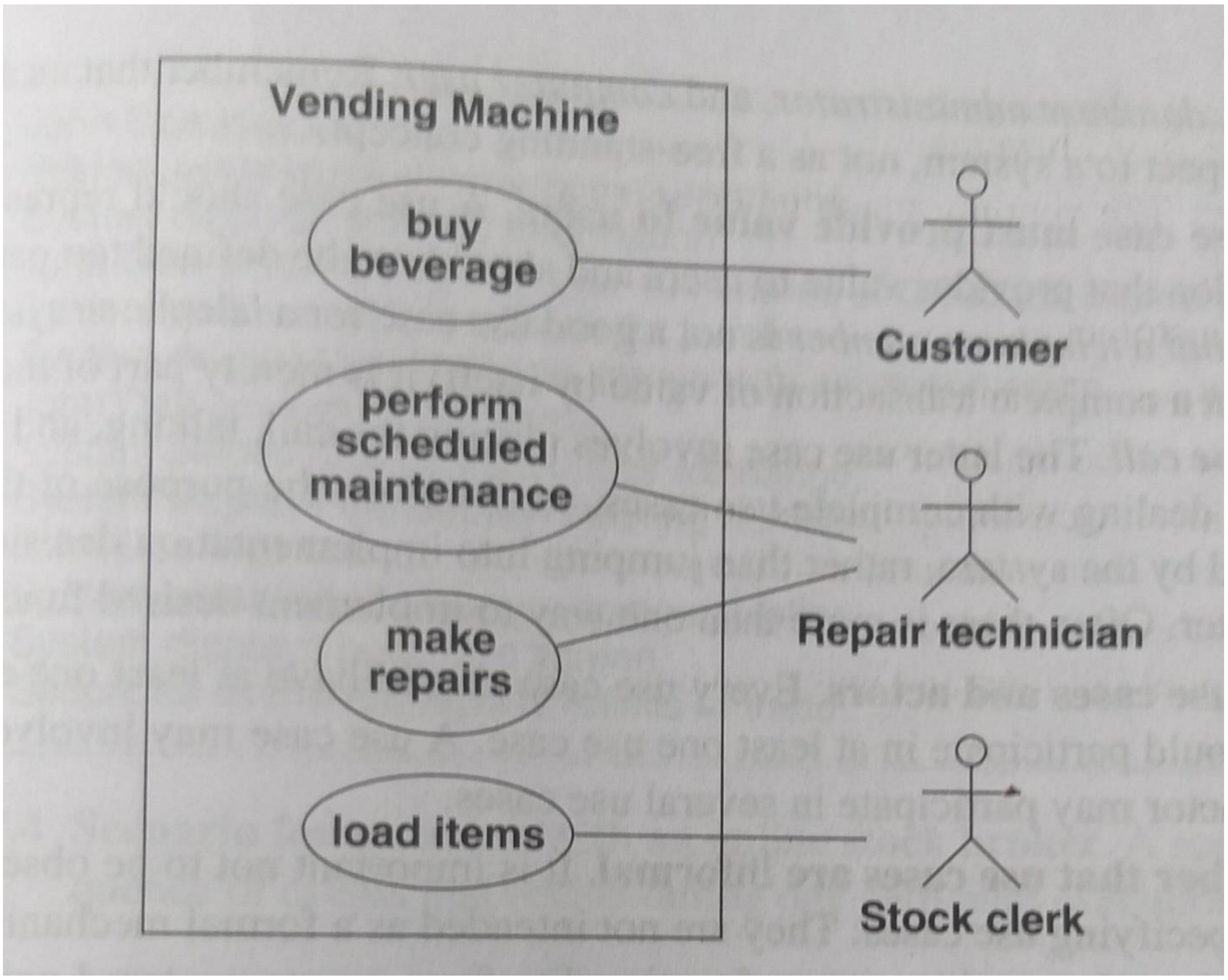
- A stick man icon denotes an actor. The name of the actor is placed below or adjacent to the stick man icon



- Solid line connect use cases to participating actors



Simple use case diagram of a Vending Machine is as follows



Some Guidelines for use case models

→ first determine System boundary

→ Ensure that actors are focused

→ Each actor should have a single, coherent purpose

→ capture different purposes with different actors.

for example, instead using a single actor for software installation, setting up a database and email sending for a computer system, Create 3-actors like System Administror, database administror and computer user.

→ Each use case must provide value to a user.

A Use-case should not be like an implementation decission. It should be represented as a Complete transaction that provide value to the user.

e.g. make a telephone call is a use case but dial a telephone number is not considered as a use-case.

→ Relate use case and actors

→ Every use case should have atleast one actor

→ Every actor should participate in atleast one use case

→ A use case may involve several actors and an actor may participate in several use cases.

→ Use cases are informal

Use cases are generally intended to identify and organize system functionality from user point of view. It is acceptable as use-cases are a bit loose at first. Details can come latter when use cases are expanded and mapped into implementation.

→ Use cases can be structured

→ for some application, individual use cases are completely distinct.

→ for large system, use cases can be built out of smaller fragments using relationships.

Use Case Relationships

→ Include relationship
→ Extend relationship
→ Generalization relationship

Include relationship

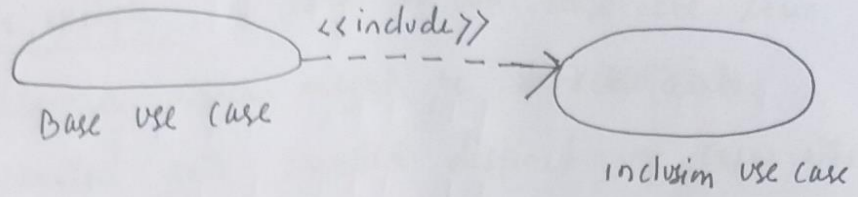
→ An include relationship is a relationship in which one use case (the base use case) includes the functionality of another use case (the inclusion use case)

→ The main reason for this is ① to reuse common actions across ~~many~~ multiple use cases.
② to simplify large use cases by splitting it into several use cases.

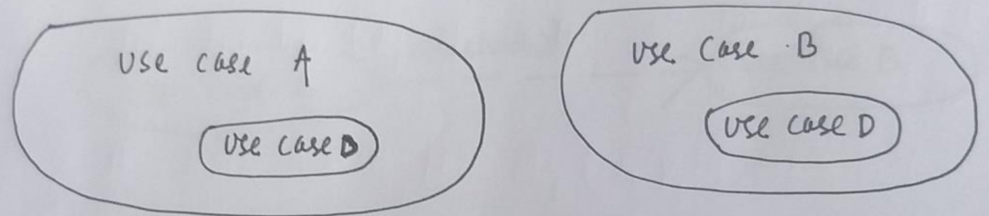
→ Basically it is the inclusion of behaviour of ~~one~~ a use case ~~f~~ into another use case.

UML Notation for include relationship

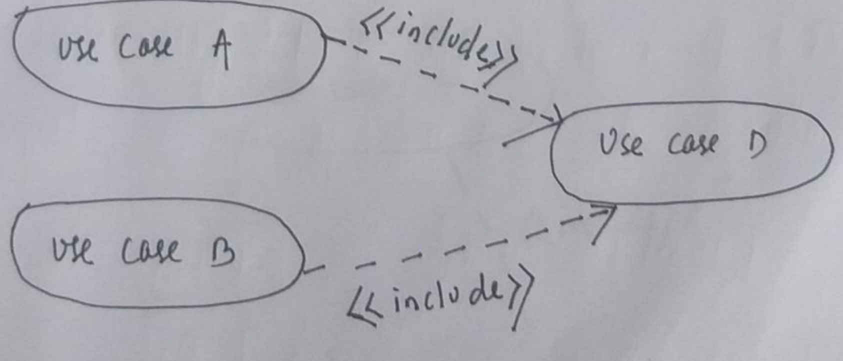
dashed arrow from source (base use case) to the inclusion use case



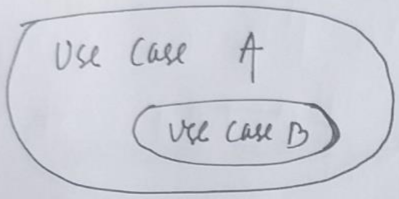
for example, when two or more use cases have some common behaviour, this common part could be extracted into a separate use case to be included back by the use cases.



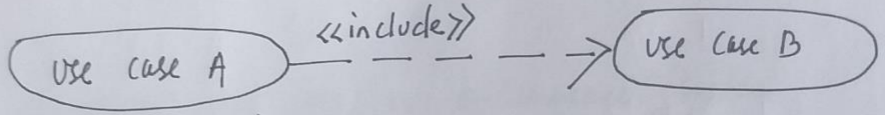
we can separate use case D and included into use case A and use case B

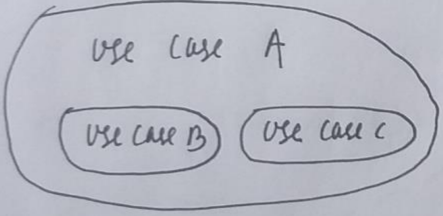


For example, a large use case could have some behaviour which might be ~~detached into~~ separated into smaller distinct use cases which are to be included to the base use case using the UML include relationship.

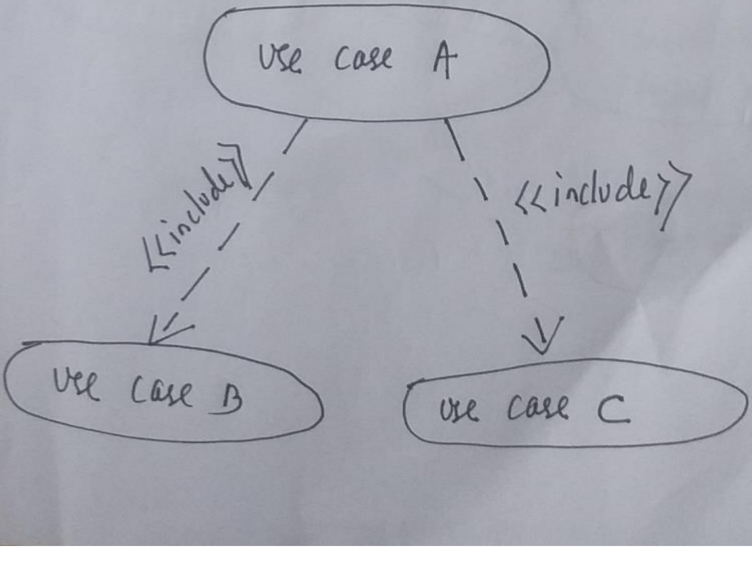


can be represented as

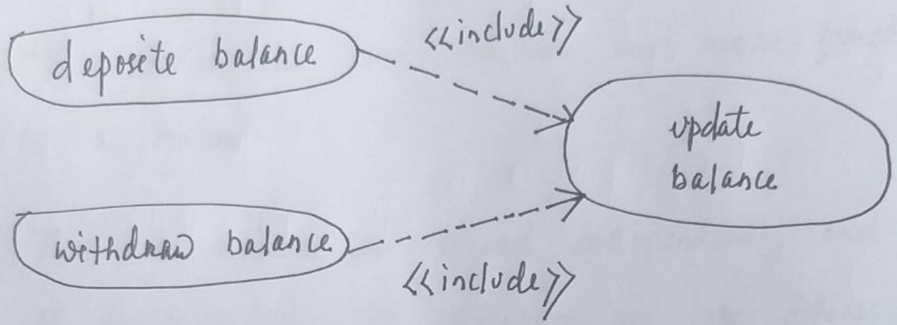




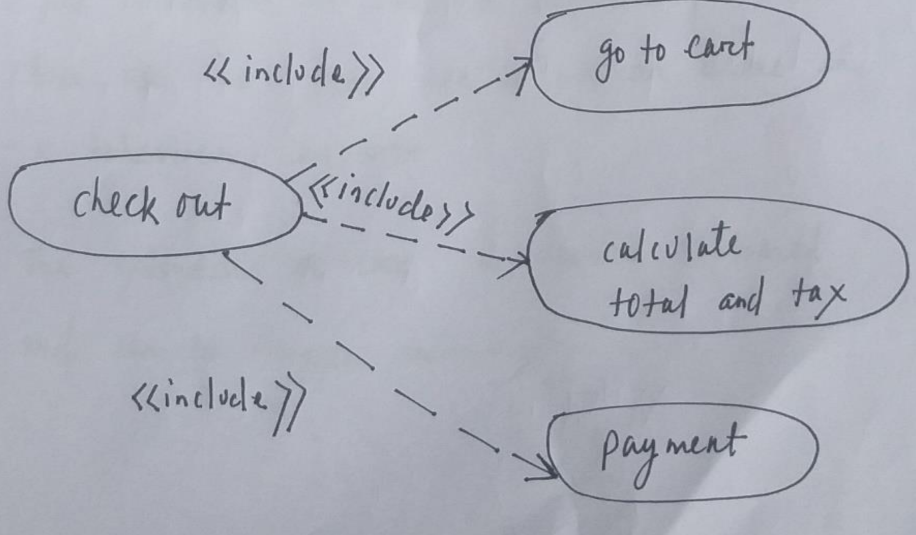
can be represented as



for example, in a banking system, use cases like deposite funds and withdraw funds includes the use case update balance



e.g. Check-out use case in e-Commerce system with include relationship



Extend Relatimship

→ The extend relatimship adds incremental behaviour or extra functimality to a use case.

→ It extends the base use case and more functimality to a system.

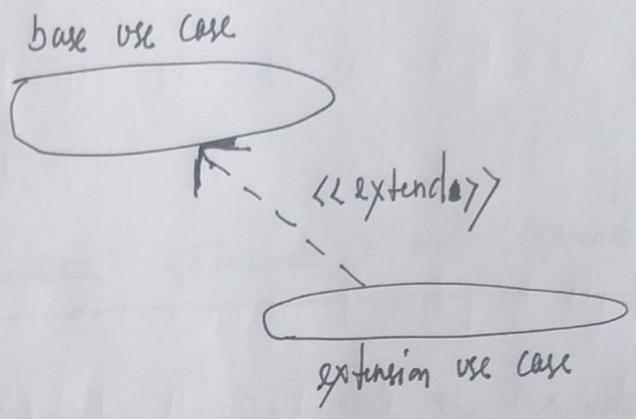
→ The base use case is defined endipendently and is meaning ful. The extension use case defines additional behaviour that can incrementally extend behaviour of the base use case

→ The extension use case is dependent on the base use case thus can't appear alone in a behaviour sequence.

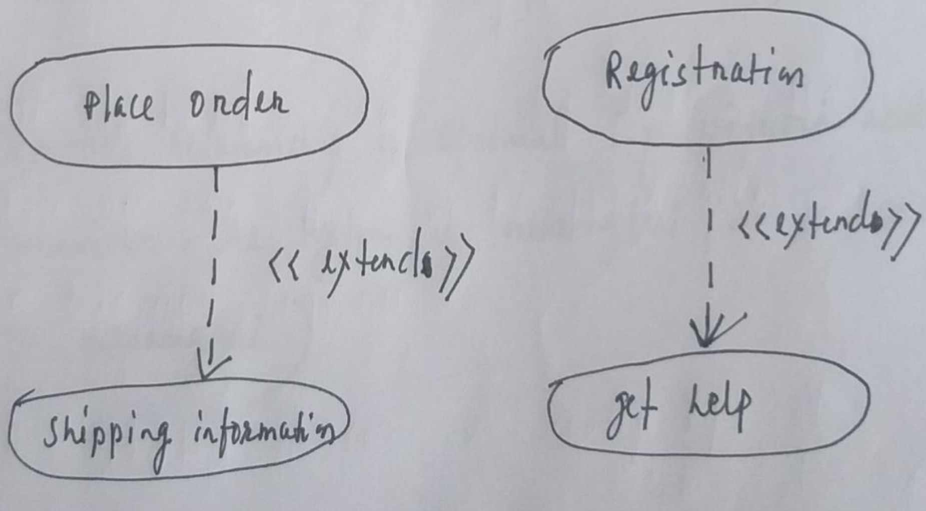
→ The extended use case is usually optimal and can be triggen conditimally.

UML Notation

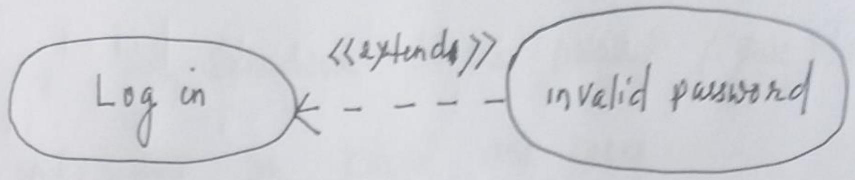
Dashed arrow from extension use case to base use case



Example



Example



~~Generalization~~

Basic diff. between <<include>> and <<extend>>

Include relationship implies that the included behaviour is a necessary part of a configured system (i.e. necessary part of a use case)

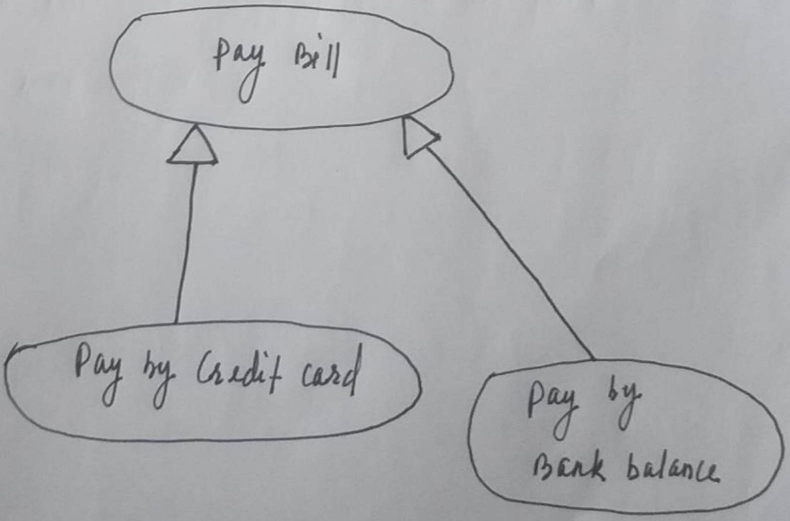
Extend relationship is optional i.e. without added behaviour, the system is meaningful (i.e. a use case is meaningful)

Generalization

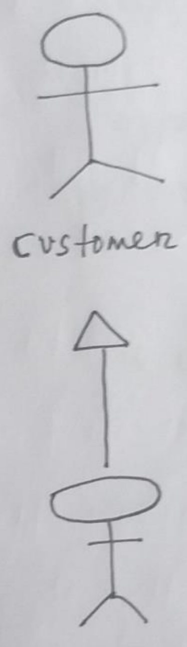
→ The behaviour of the parent (base) use case enheritted by child use cases.

→ A parent use case represent a general behaviour sequence. child use cases ~~specialized~~ specialize the parent by inserting additional steps.

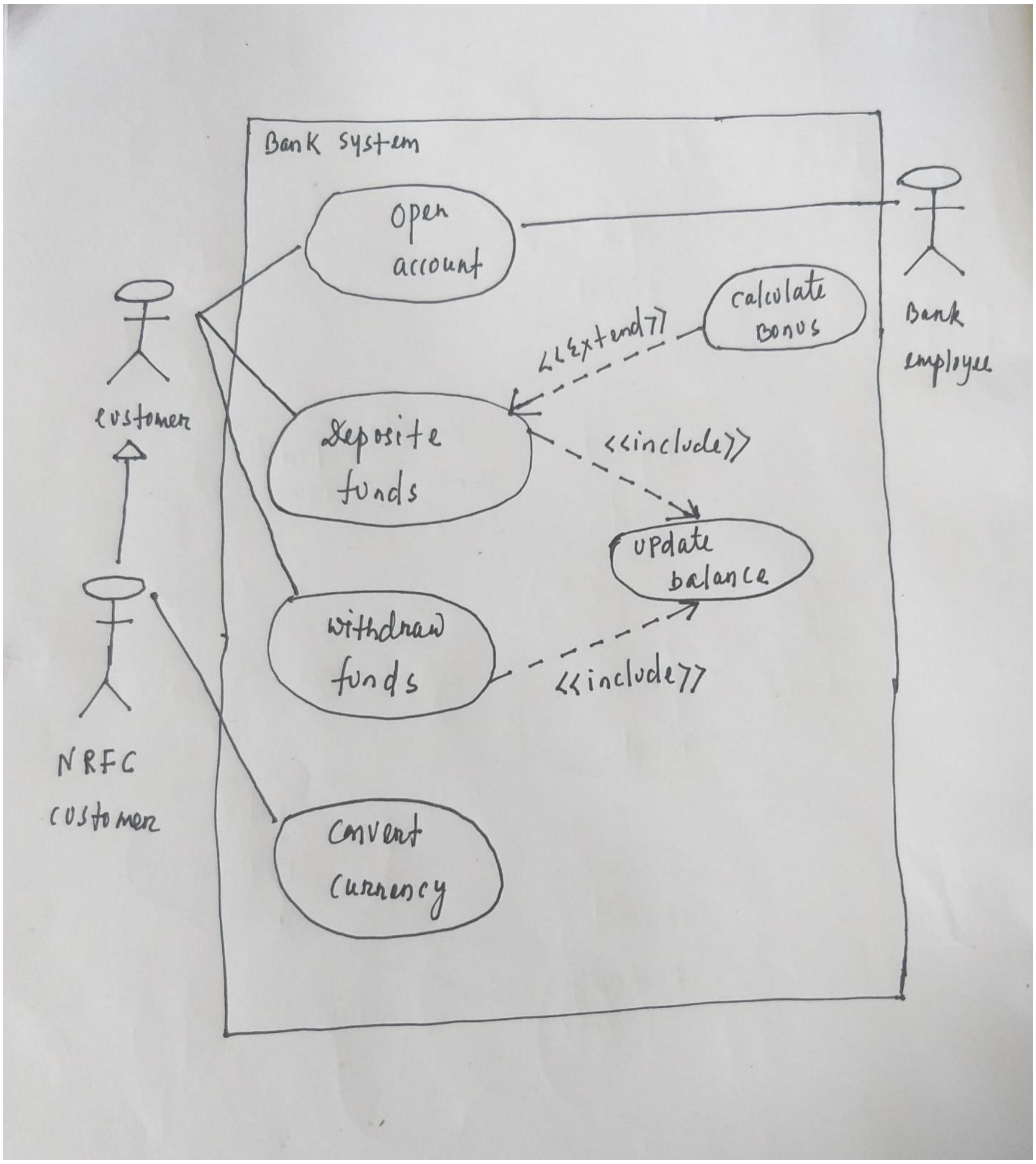
→ Common behaviour of the base use ~~cases~~ Case ~~are shared~~ (or parent use case) shared by child use cases and child use cases adds their own behaviours specific to the requirements



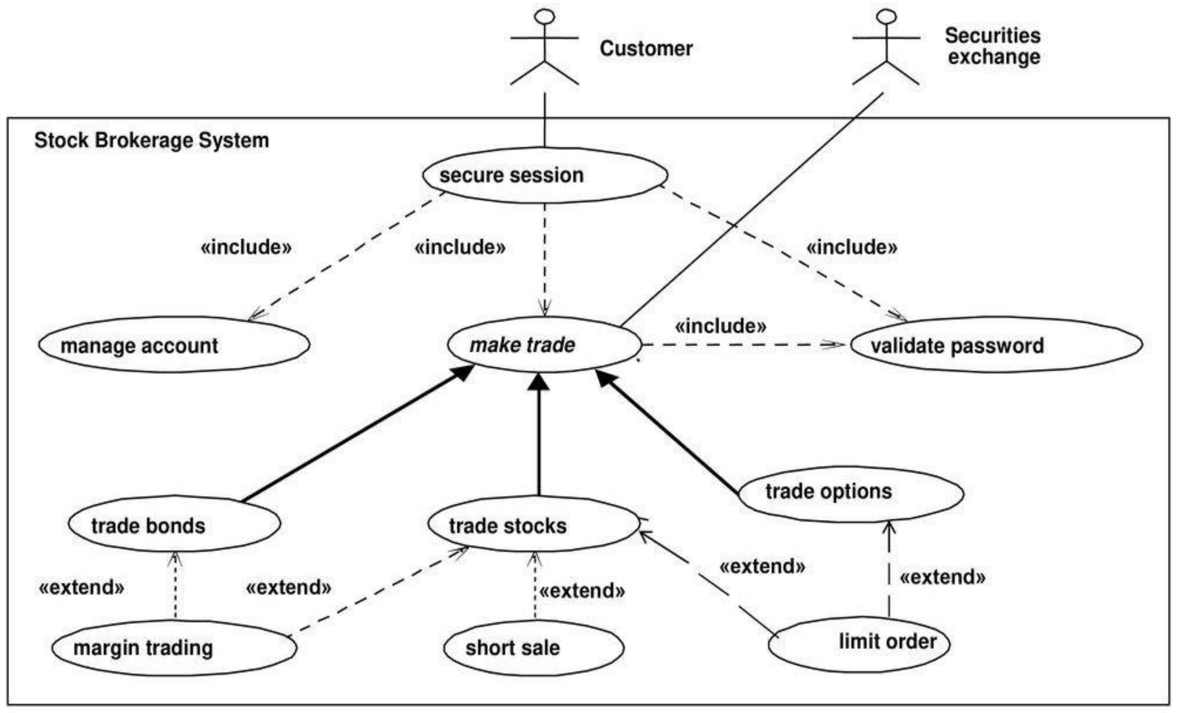
Note: In the same way as in use case generalization, Generalization of actors can also be possible. The child inherits all the use cases of parent



The following is the Use Case diagram of a **Banking system** with usual assumptions.



The following is the **Use Case diagram** of a **Stock Brokerage System** (***See Book*** **(Ch-8)** ***that I have provided***)



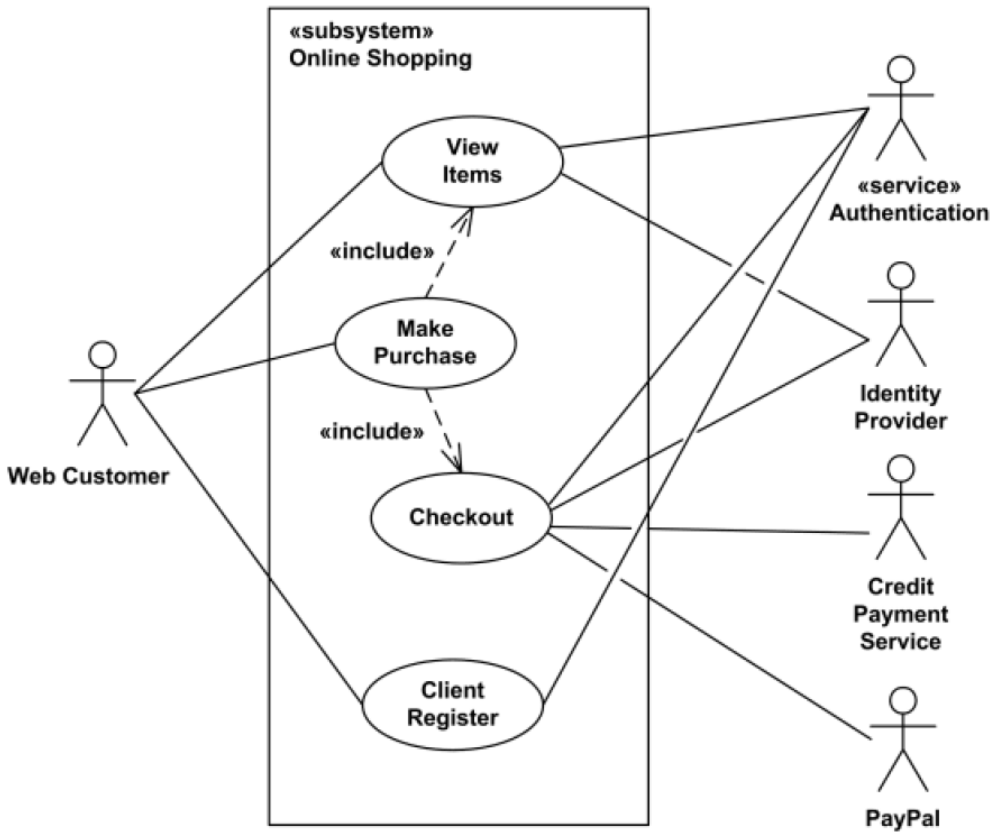
Use Case Diagram of Online Shopping is as follows.

Web Customer actor uses some web site to make purchases online. Top level use cases are **View Items, Make Purchase** and **Client Register**.

View Items use case could be used by customer if customer only wants to find and see some products. This use case could also be used as a part of **Make Purchase** use case.

Client Register use case allows customer to register on the web site, for example to get some coupons or be invited to private sales.

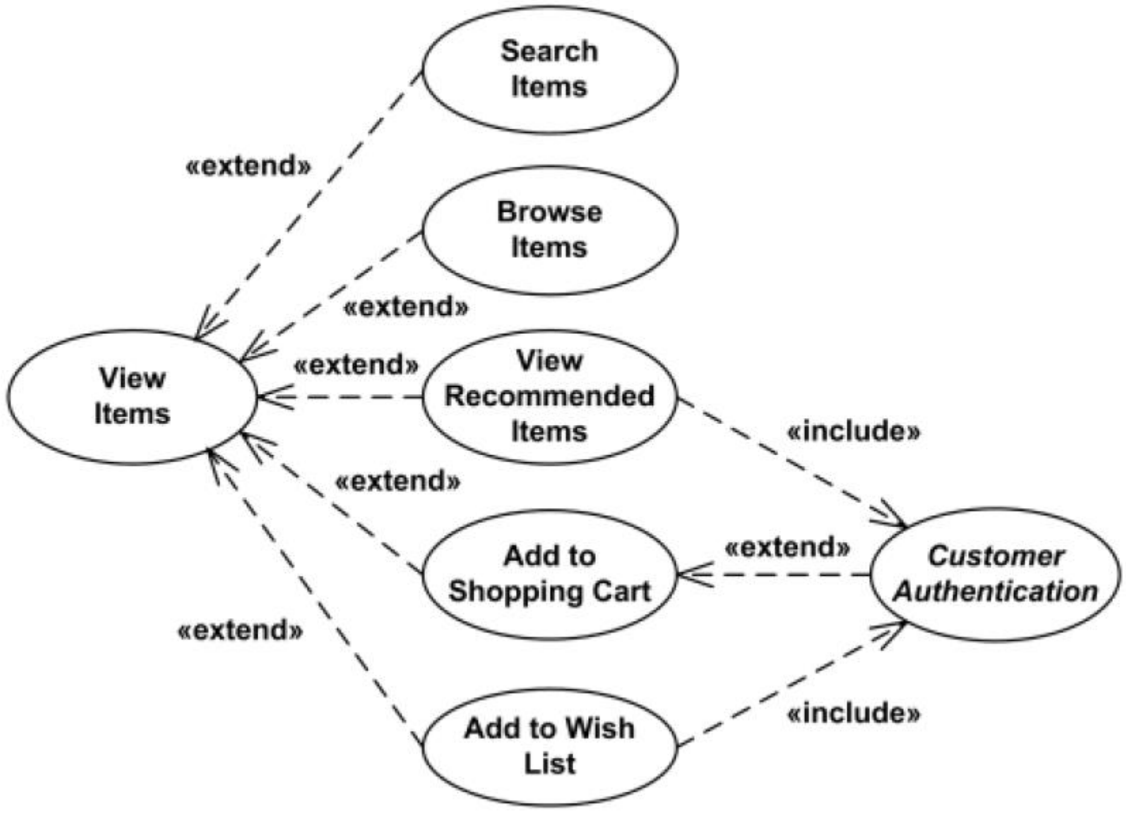
The **Checkout** use case is an included use case not available by itself - **checkout** is part of **making purchase** use case**.**



View Items use case is extended by several optional use cases - customer may search for items, browse catalog, View items recommended for him/her, add items to shopping cart or wish list. All these use cases are extending use cases because they provide some optional functions allowing customer to find item.

Customer Authentication use case is included in **View Recommended Items** and **Add to Wish List** because both require customer to be authenticated. At the same time, item could be added to the shopping cart without user authentication.

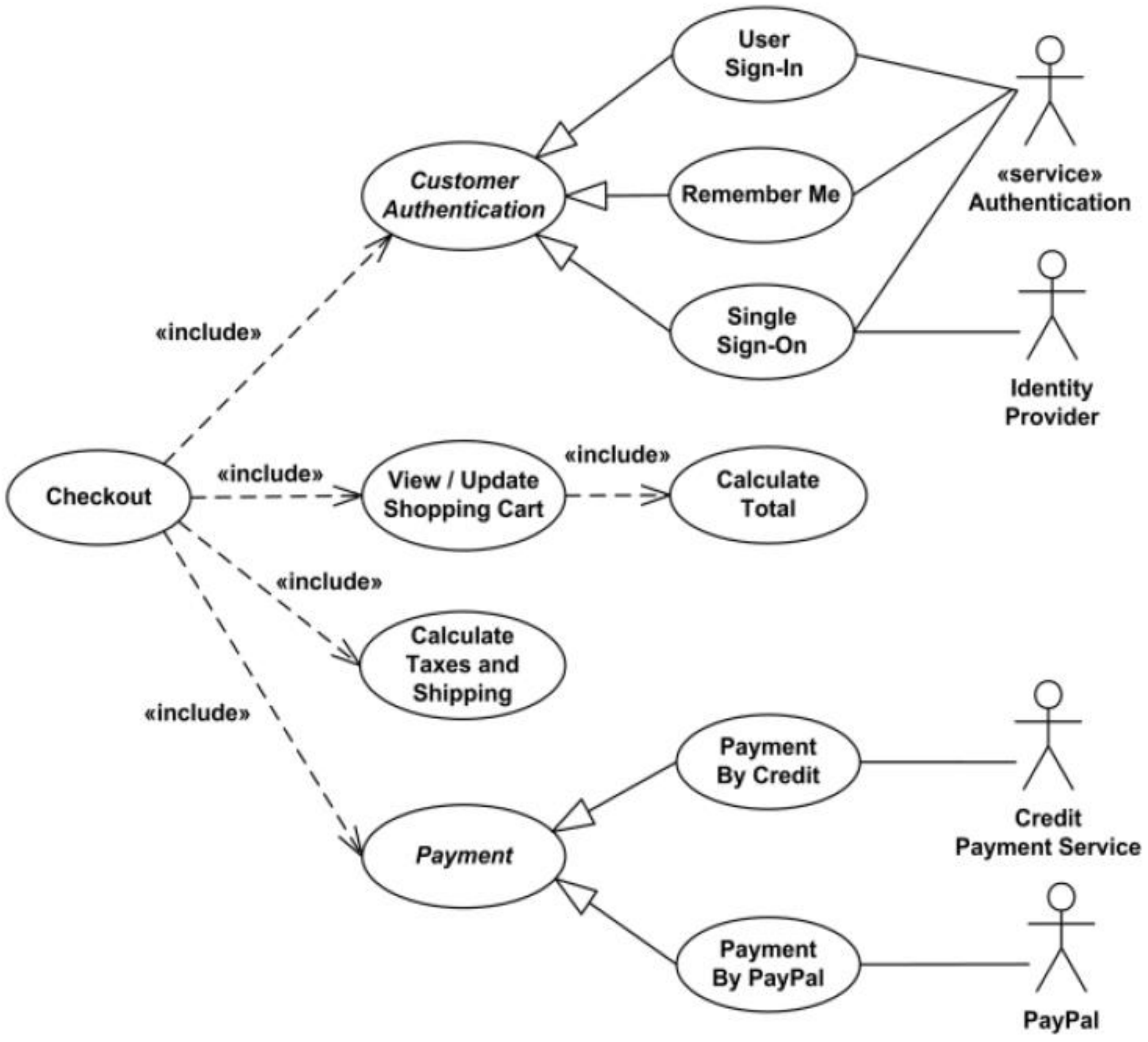
View Items use case can be modeled as follows



Checkout use case includes several required uses cases. Web customer should be authenticated. It could be done through user login page, user authentication cookie ("Remember me") or Single Sign-On (SSO). Web site authentication service is used in all these use cases, while SSO also requires participation of external identity provider.

Checkout use case also includes **Payment** use case which could be done either by using credit card and external credit payment service or with PayPal.

The following diagram shows **Checkout**, **Authentication** and **Payment** use cases.


User Sign-In
Customer Authentication
Remember Me
Single Sign-On
«service» Authentication
Identity Provider
«include»
Checkout
«include»
View / Update Shopping Cart
«include»
Calculate Total
«include»
Calculate Taxes and Shipping
«include»
Payment By Credit
Credit Payment Service
Payment
Payment By PayPal
PayPal