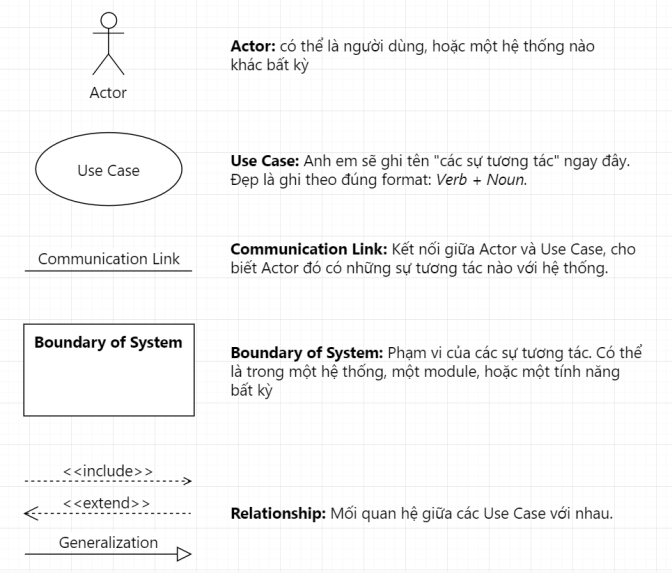
**2. Các thành phần của Use Case Diagram**

**2.1. Actor, Use Case, Communication Link và Boundary**

Cũng không có gì quá phức tạp, Use Case Diagram gồm 5 thành phần chính:

* Actor
* Use Case
* Communication Link
* Boundary of System
* Relationships.

[](https://thinhnotes.com/chuyen-nghe-ba/use-case-diagram-va-5-sai-lam-thuong-gap/attachment/cac-thanh-phan-cua-usecase-thinhnotes/)

**Actor** thì có thể là Người dùng, hoặc một System nào đó. Vì UML quy định Actor là hình thằng người nên có thể anh em sẽ nhầm lẫn chỗ đó phải là người dùng nhưng hổng phải.

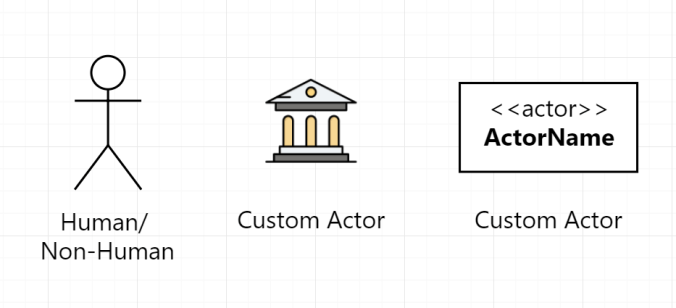
Một số câu hỏi anh em có thể tự lẩm bẩm trong đầu để xác định Actor như sau:

* Ai là người sử dụng hệ thống?
* Ai sẽ là người Admin của hệ thống (tức người cài đặt, quản lý, bảo trì… hệ thống)?
* Hệ thống này có được sử dụng bởi bất kỳ một hệ thống nào khác không? ***(\*)***
* Hệ thống lưu trữ dữ liệu, vậy ai là người input dữ liệu vào hệ thống?
* Hệ thống lưu trữ dữ liệu, vậy ai là người cần những dữ liệu output?

Ở mục (\*), mình muốn highlight cho anh em chỗ này. Không phải giải pháp/ phần mềm nào làm ra đều được sử dụng bởi con người. Có những phần mềm làm ra, để cho… phần mềm khác sử dụng.

Chẳng hạn như làm các services. Mình có một anh bạn làm BA, giải pháp mà ảnh cùng đồng bọn làm ra là 1 services không được dùng bởi con người, mà được dùng bởi một hệ thống khác để xác thực người dùng.

Ký hiệu của Actor chủ yếu là hình thằng người, nhưng để Diagram thêm phong phú, đa dạng thì anh em có thể sử dụng các hình dưới đây, miễn có ghi chú rõ ràng là được.



**Các ký hiệu thể hiện Actor.**

Còn **Use Case** là anh em sẽ thể hiện dưới dạng hình Oval, thể hiện sự tương tác giữa các Actor và hệ thống.

**Communication Link** thể hiện sự tương tác giữa Actor nào với System. Nối giữa Actor với Use Case.

**Boundary of System** là phạm vi mà Use Case xảy ra. Ví dụ trong hệ thống CRM, phạm vi có thể là từng cụm tính năng lớn như Quản lý khách hàng, Quản lý đơn hàng, hoặc cả một module lớn như Quản lý bán hàng.

…

Ô kê nãy giờ dễ ẹc, mấy cái này nhìn sơ qua là anh em biết ngay cái một.

Cái cuối cùng mới chính là cái mà mình tin là nhiều anh em vẫn còn rất dễ lộn, đó là **Relationship.**

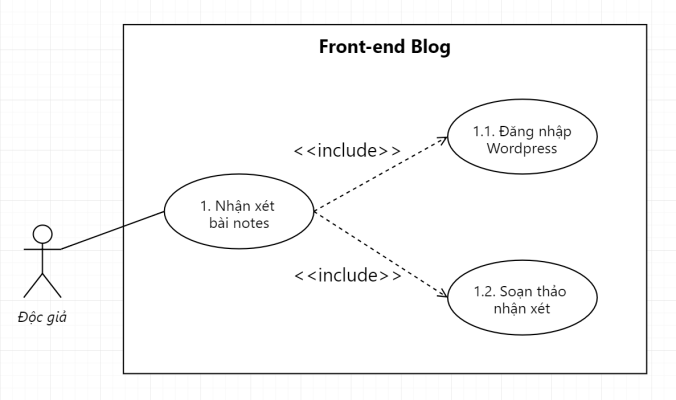
**2.2. Relationship**

Relationship gồm 3 loại: **Include**, **Extend**, và **Generalization**.

**a) Include**

Include nghĩa là mối quan hệ**bắt buộc phải có** giữa các Use Case với nhau.

Xét về nghĩa, Include nghĩa là bao gồm, tức nếu Use Case A có mối quan hệ include Use Case B, thì nghĩa là: **Use Case A bao gồm Use Case B.** Để Use Case A xảy ra, thì Use Case B phải đạt được.



**Ví dụ về Include trong Use Case**

Xét ví dụ trên, chúng ta có Use Case: Nhận xét bài notes. Use Case này include 2 Use Case khác là: Đăng nhập WordPress và Soạn thảo nhận xét.

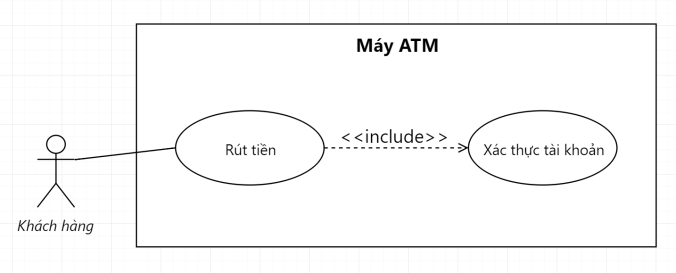
Rõ ràng anh em thấy: để nhận xét được một bài viết, anh em cần phải đăng nhập vào 1 tài khoản nào đó, để blog nhận diện anh em là ai, tên gì, quê quán, giai gái ra sao.

Ví dụ ở blog mình là anh em sẽ cần đăng nhập vào tài khoản WordPress. Sau khi đăng nhập xong, anh em phải soạn thảo nhận xét, tức là gõ nhận xét, chỉnh sửa, xóa tới xóa lui. Sau khi viết xong nhận xét, anh em sẽ bấm nút Submit để hoàn thành chẳng hạn.

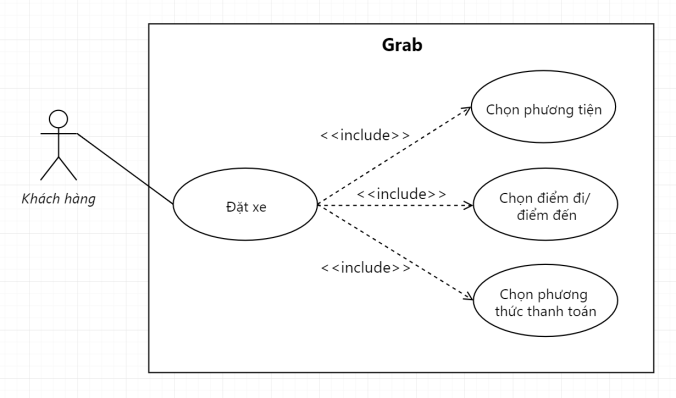
Chỉ khi nào xong 2 bước trên (đăng nhập và soạn thảo nhận xét), thì anh em mới có thể xong bước Nhận xét bài notes được.

Hay nói cách khác **để *Use Case: Nhận xét bài notes xảy ra*, thì *Use Case: Đăng nhập WordPress* và *Use Case: Soạn thảo nhận xét* phải bắt buộc hoàn thành trước tiên.**

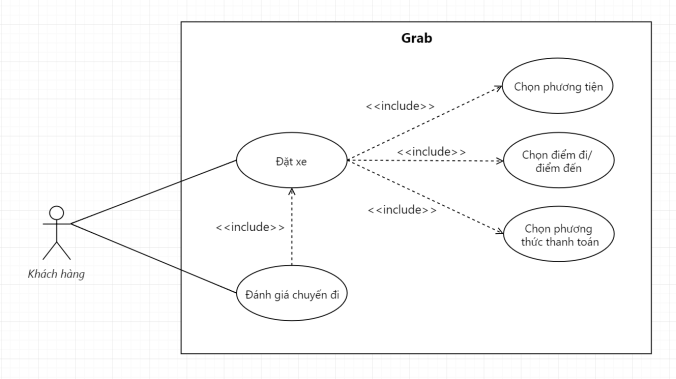
Đó chính là mối quan hệ Include. Anh em xem tiếp 1 số ví dụ dưới cho dễ hình dung nhé.



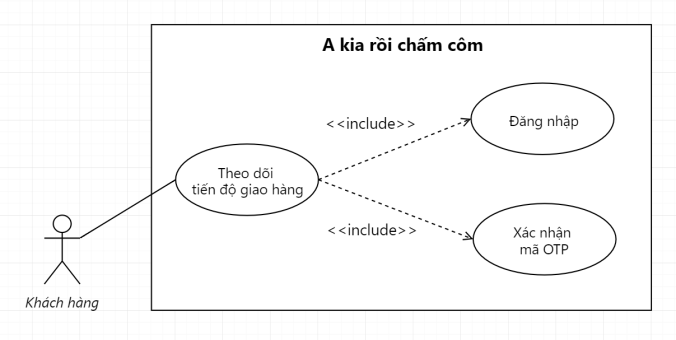
**Muốn rút được tiền thì đầu tiên khách hàng phải xác thực tài khoản đi cái đã.**



**Muốn đặt được xe thì phải hoàn thành được 3 bước này rồi hệ thống mới cho đặt.**



**Hoặc khách hàng muốn đánh giá được chuyến đi thì trước đó họ phải đặt xe cái đã.**



**Hoặc tương tự là Use Case thể hiện tính năng Theo dõi tiến độ giao hàng trên một trang e-Commerce bất kỳ.**

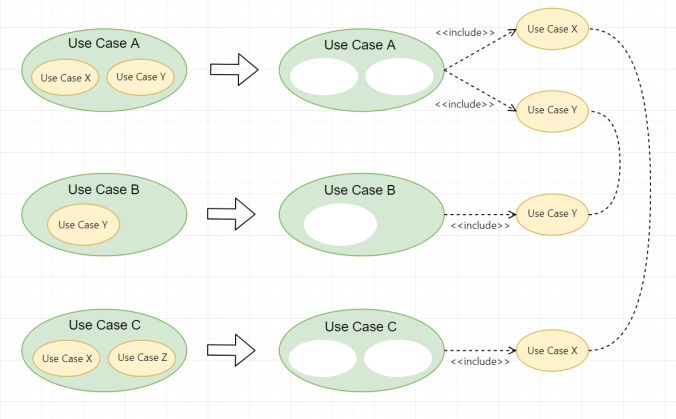
**Một số điểm cần chú ý khi vẽ Include cho Use Case**

Thực sự không có quy tắc nào rõ ràng cho việc khi nào cần tách Use Case ra thành các Use Case nhỏ và cho nó một mối quan hệ Include cả.

Việc tách hay không tách phụ thuộc duy nhất vào người vẽ. Và lý do lớn nhất để mối quan hệ Include ra đời là giúp cho các Use Case của chúng ta**DỄ QUẢN LÝ** hơn; làm cho Use Case Diagram trông có vẻ nguy hiểm hơn mà thôi

Và anh em chỉ nên tách Use Case khi nó có ***độ phức tạp lớn*** và những thứ tách ra được có thể được ***tận dụng ở các Use Case sau này***.

Độ phức tạp lớn thì khi tách ra mình mới có được những Use Case vừa phải, đủ để diễn đạt dễ hiểu cho các stakeholders. Còn tận dụng được ở các Use Case sau là sao?



**Sử dụng Include như thế nào cho hợp lý?**

Ví dụ: Use Case A gồm 2 Use Case nhỏ bên trong là X và Y. Do đó Use Case A được tách thành Use Case X và Use Case Y.

Tương tự, Use Case B gồm Use Case Y bên trong, nên được tách thành Use Case Y.

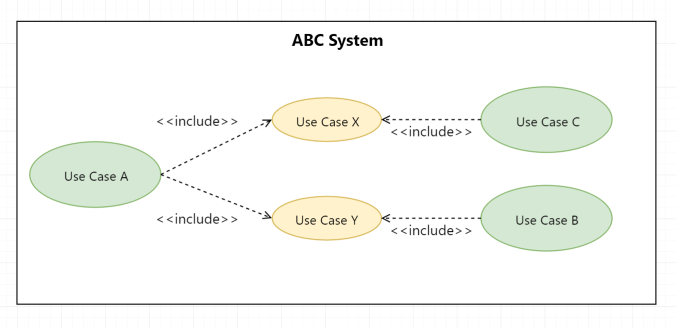
Nhưng, Use Case C gồm Use Case X và Use Case Z bên trong, nhưng chỉ có Use Case X là được tách ra cho mối quan hệ Include. Vì có thể Use Case Z “không đáng” để tách ra thành một Use Case nhỏ hơn.

Chúng ta tách Use Case X từ Use Case A để Use Case C có thể tận dụng được mà không cần vẽ lại. Tương tự, tách Use Cas Y từ Use Case B để Use Case A có thể tận dụng mà cũng không cần vẽ lại.

Điều này giúp Use Case Diagram của chúng ta trở nên **chặt chẽ**, **logic** và**gọn nhẹ** hơn rất nhiều.

Còn Use Case Z, vì nó không được “dùng lại” ở một Use Case bất kỳ nào sau đó, nên người vẽ có thể cân nhắc có tách nó ra hay không!

Nếu Use Case đó đủ lớn và khá là high-level, thì có lẽ chúng ta nên tách. Còn nếu ngược lại, Use Case đã rõ ràng, là một requirement từ phía User cụ thể thì không đáng để anh em tách nó ra thành một Use Case nhỏ, chỉ làm hình thêm thêm rối mà thôi.



**Khi tách Use Case ra thành các Use Case nhỏ để tận dụng mối quan hệ Include, anh em hãy nhớ 2 thứ: độ lớn của Use Case và khả năng tái sử dụng lại của nó.**

Còn cách vẽ thì anh em cứ nhớ là include tới thằng nào thì dấu mũi tên hướng tới thằng đó nhé anh em. Nhớ để qua phần Extend cho khỏi lộn.

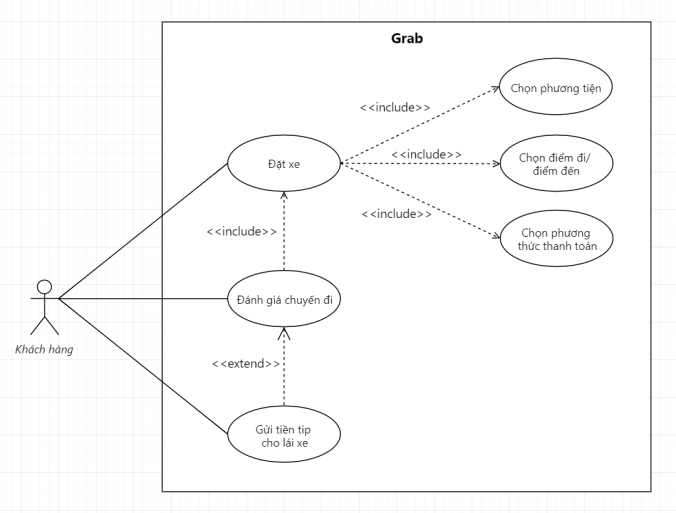
**b) Extend**

Extend là mối quan hệ **mở rộng** giữa các Use Case với nhau.

Nếu Include là mối quan hệ bắt buộc, thì Extend là một mối quan hệ **không bắt buộc.**Nó thể hiện mối quan hệ có thể có hoặc có thể không giữa các Use Case với nhau.

Một Use Case B là extend của Use Case A thì có nghĩa Use Case B chỉ là một thứ optional, và chỉ xảy ra trong một hoàn cảnh cụ thể nào đó.

Lấy ví dụ Grab phía trên, anh em sẽ dễ dàng có được một mối quan hệ Extend như sau.



**Ví dụ về mối quan hệ Extend giữa các Use Case**

Trong trường hợp này, ***Use Case: Gửi tiền tip cho lái xe*** là một Use Case có mối quan hệ Extend với ***Use Case: Đánh giá chuyến đi***. Tức, Use Case: Gửi tiền tip cho lái xe chỉ là một Use Case có thể xảy ra, hoặc không; và nó **liên quan đến** Use Case: Đánh giá chuyến đi, chứ không phải bất kỳ một Use Case nào khác.

À…à…Nhắc tới lúc có lúc không, tức là nhắc tới điều kiện xảy ra.

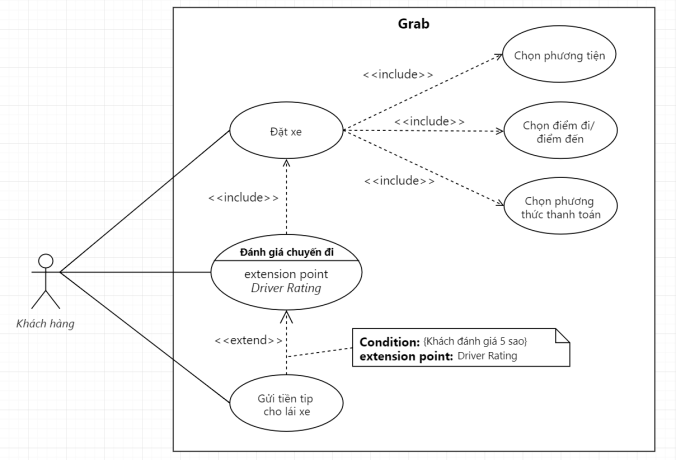
Anh em có thể thể hiện rõ ý chỗ này bằng một thứ luôn đi kèm với Extend, đó là **Extension Point**

Extension Point nôm na là điều kiện mà Use Case có mối quan hệ Extend sẽ xảy ra. Còn để sát nghĩa thì anh em có thể hiểu chữ Point ở đây nghĩa là điểm dữ liệu thể hiện sự khác biệt.

Tức nếu dữ liệu này là A thì Use Case không xảy ra, nhưng nếu dữ liệu này là B thì Use Case sẽ xảy ra.

//Theo mình nhớ là hình như anh em chỉ có thể gửi tiền tip cho tài xế, nếu cuốc xe đó anh em chấm họ maximum là 5 sao.//

Vậy thì anh em sẽ vẽ Use Case Diagram chỗ đó như sau.



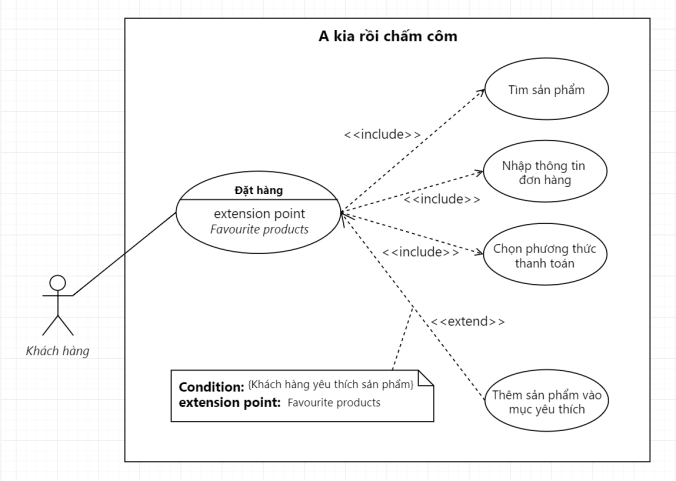
**Use Case Diagram có thể hiện rõ khi nào thì mối quan hệ Extend diễn ra.**

Extension Point ở đây là dữ liệu Driver Rating. Nếu Driver Rating đạt giá trị 5 sao, thì Use Case: Gửi tiền tip cho lái xe sẽ xảy ra, và hoàn toàn optional, tùy thuộc vào khách hàng.

Và nó liên quan mật thiết đến Use Case: Đánh giá chuyến đi, là một phần mở rộng của Use Case: Đánh giá chuyến đi.

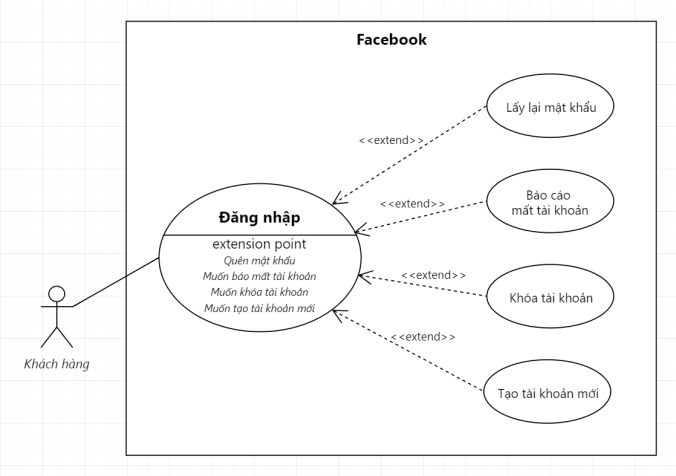
Extension Point không nhất thiết phải là một dữ liệu nào đó trên hệ thống, mà có thể là một “điều kiện” bất kỳ, miễn là nó thể hiện được trường hợp cụ thể mà Use Case sẽ xảy ra.

Ở một ví dụ khác.



**Mối quan hệ Extend trong Use Case Diagram của A kia rồi chấm côm.**

Còn nếu Use Case có quá nhiều mối quan hệ Extend, làm cho Diagram nhìn rối bời quá, anh em có thể bỏ luôn phần comment của Extension Point luôn cũng được.



**Vẽ vầy cũng hông sao.**

**c) Generalization**

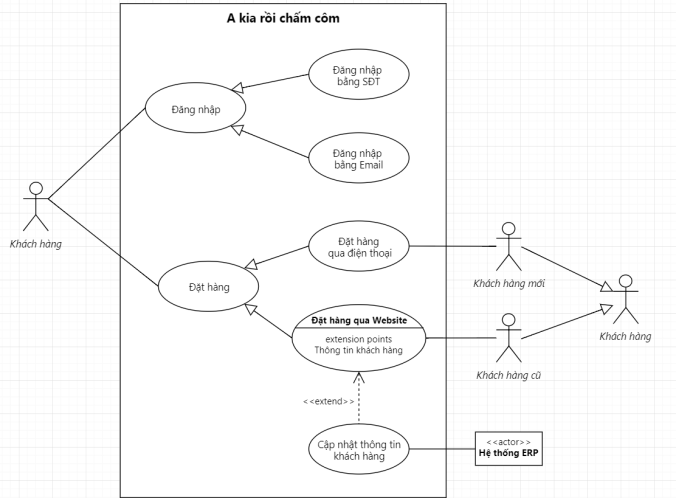
Generalization đơn giản là quan hệ cha con giữa các Use Case với nhau. Nhưng khác biệt với Include và Extend là nó còn được dùng để thể hiện mối quan hệ giữa các… Actor với nhau.

Đầu tiên là mối quan hệ cha-con giữa các Use Case. Ví dụ:

* Đăng nhập thì có thể đăng nhập qua số điện thoại, hoặc đăng nhập qua email.
* Đặt hàng thì có đặt hàng qua điện thoại, hoặc đặt hàng qua website.
* Thanh toán thì thanh toán qua thẻ ATM, qua thẻ thanh toán quốc tế, hoặc qua ví điện tử.
* Hoặc tìm kiếm thì có thể tìm kiếm bằng từ khóa, hoặc tìm kiếm theo nhóm sản phẩm.

Hoặc mối quan hệ cha-con giữa các Actor. Ví dụ:

* Khách hàng gồm khách hàng cũ và khách hàng mới
* Hoặc Vendor thì có thể gồm Retailers và Wholesalers.



**Ví dụ về quan hệ cha-con (Generalization) trong Use Case.**

Nhìn chung, Generalization giúp anh em **thể hiện rõ hơn** các yêu cầu bằng việc gom nhóm các Use Case lại theo quan hệ cha-con. Cá nhân mình thì rất ít khi vẽ relationship này, chủ yếu chỉ dùng Include và Extend là chính.

Còn một điểm nữa là Generalization có **tính kế thừa**. Tức thằng cha có gì thì thằng con có cái đó, kể cả Use Case hay Actor.

Ví dụ Use Case A có include đến Use Case B và C. Thì Use Case A’ là con của Use Case A cũng sẽ có mối quan hệ Include đến Use Case B và C, mặc dù không được thể hiện trên hình.

…

Ô kê, vậy là xong phần Relationship – một trong những phần chuối nhất, dễ lộn nhất trong Use Case. Hi vọng những ví dụ trên giúp anh em hiểu được cụ thể như thế nào là Include, Extend và Generalization trong một Use Case Diagram