Object-Oriented Analysis and Design using JAVA

B.Tech (CSE/IT) 5th SEM
2020-2021

Lecture-17 Relationships –Use case

Introduction

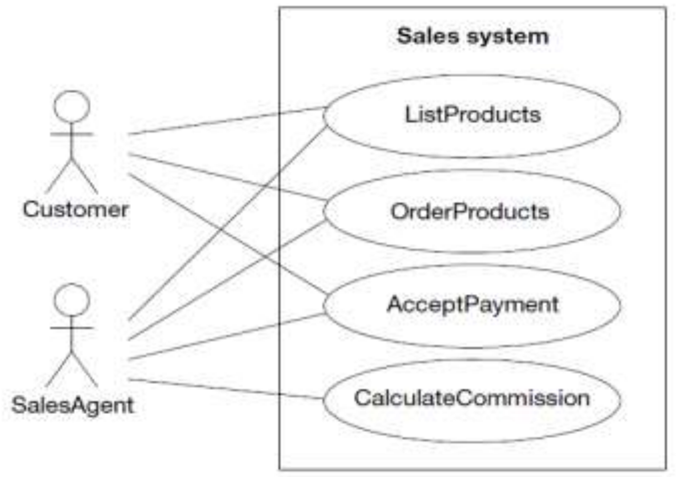
We will discuss the relationships that are possible between actors and actors, and between use cases and use cases. These relationships are as follows.

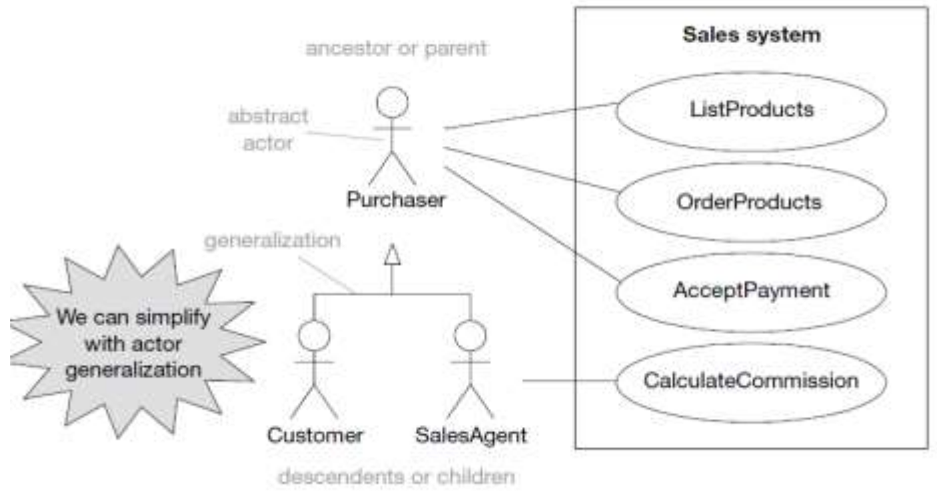
- **Actor generalization** – a generalization relationship between a more general actor and a more specific actor.
- **Use case generalization** – a generalization relationship between a more general use case and a more specific use case.
- **«include»** – a relationship between use cases that lets one use case include behavior from another.
- **«extend»** – a relationship between use cases that lets one use case extend its behavior with one or more behavior fragments from another.

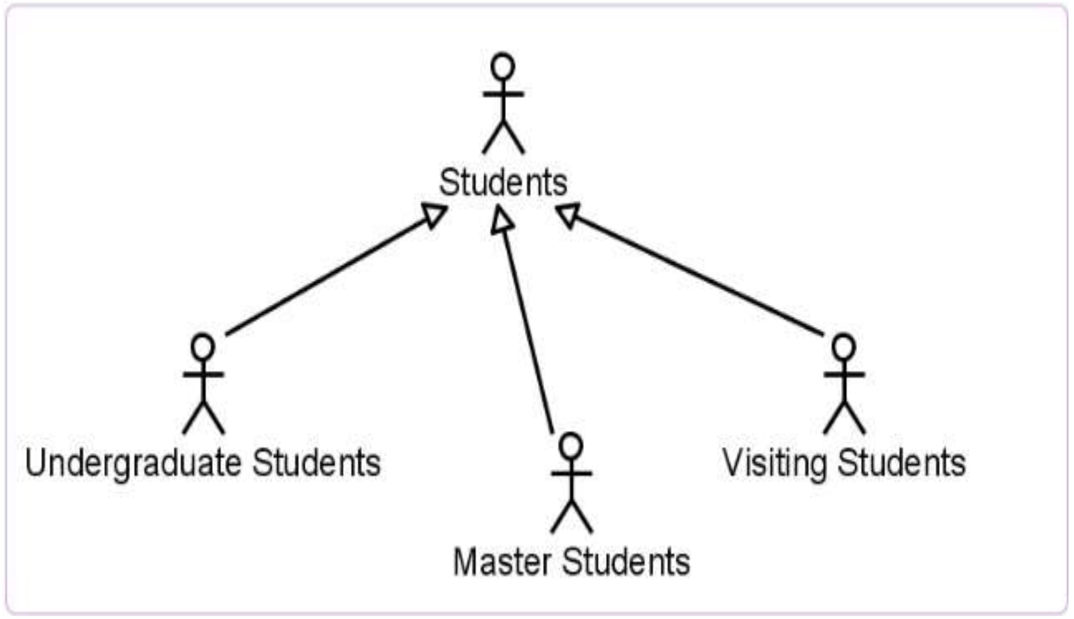
It is very important to keep all models as simple as possible, so these relationships should be used with discretion and only where they improve the overall clarity of the use case model. It is easy to go overboard with «include» and «extend» in particular, but you must avoid this.

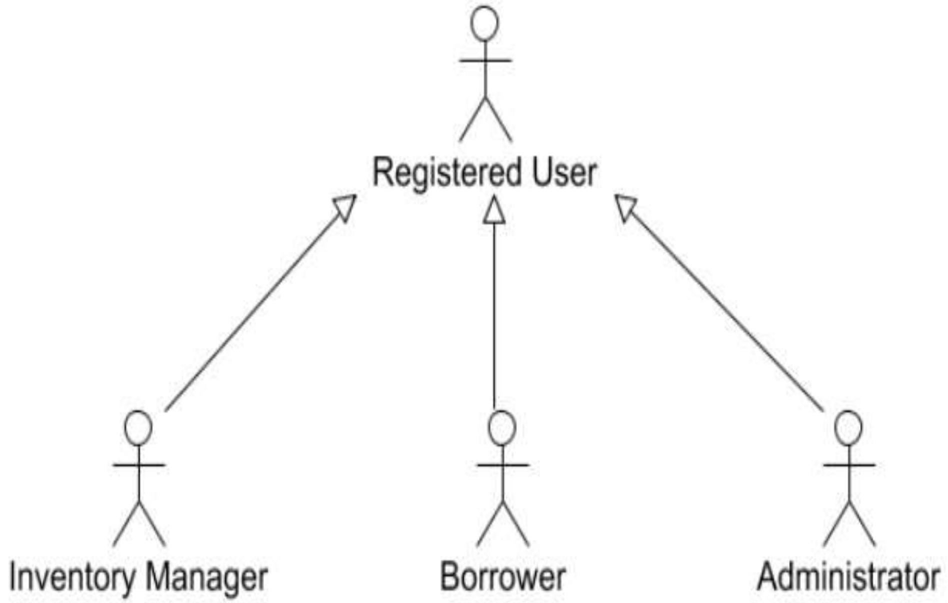
Actor generalization?

In the Figure, you can see that there is quite a lot of commonality between the two actors, Customer and SalesAgent, in the way that they interact with the Sales system (here, the SalesAgent can handle a sale on behalf of a Customer). Both actors trigger the use cases ListProducts, OrderProducts, and AcceptPayment. In fact, the only difference between the two actors is that the SalesAgent also triggers the CalculateCommission use case. Apart from the fact that this similarity in behavior gives lots of crossed lines on the diagram, it seems to indicate that there is some common actor behavior that could be factored out into a more *generalized* actor.









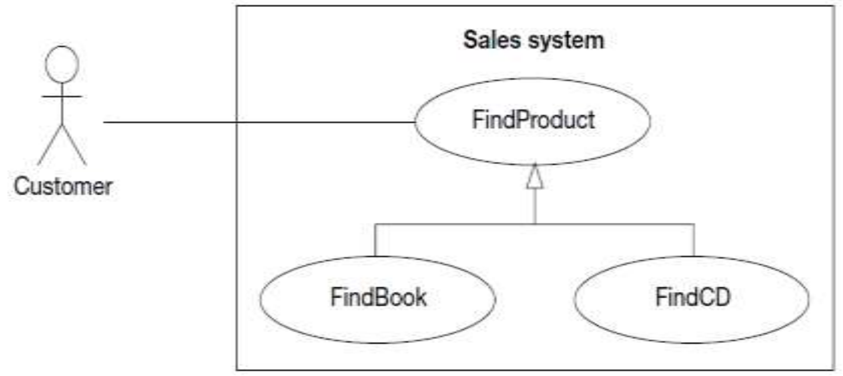
Referred sources mentioned in Key References

Use case generalization?

Use case generalization is used when you have one or more use cases that are really specializations of a more general case. Just like actor generalization, you should only use this when it simplifies your use case diagrams. In use case generalization, the child use cases represent more specific forms of the parent. The children may:

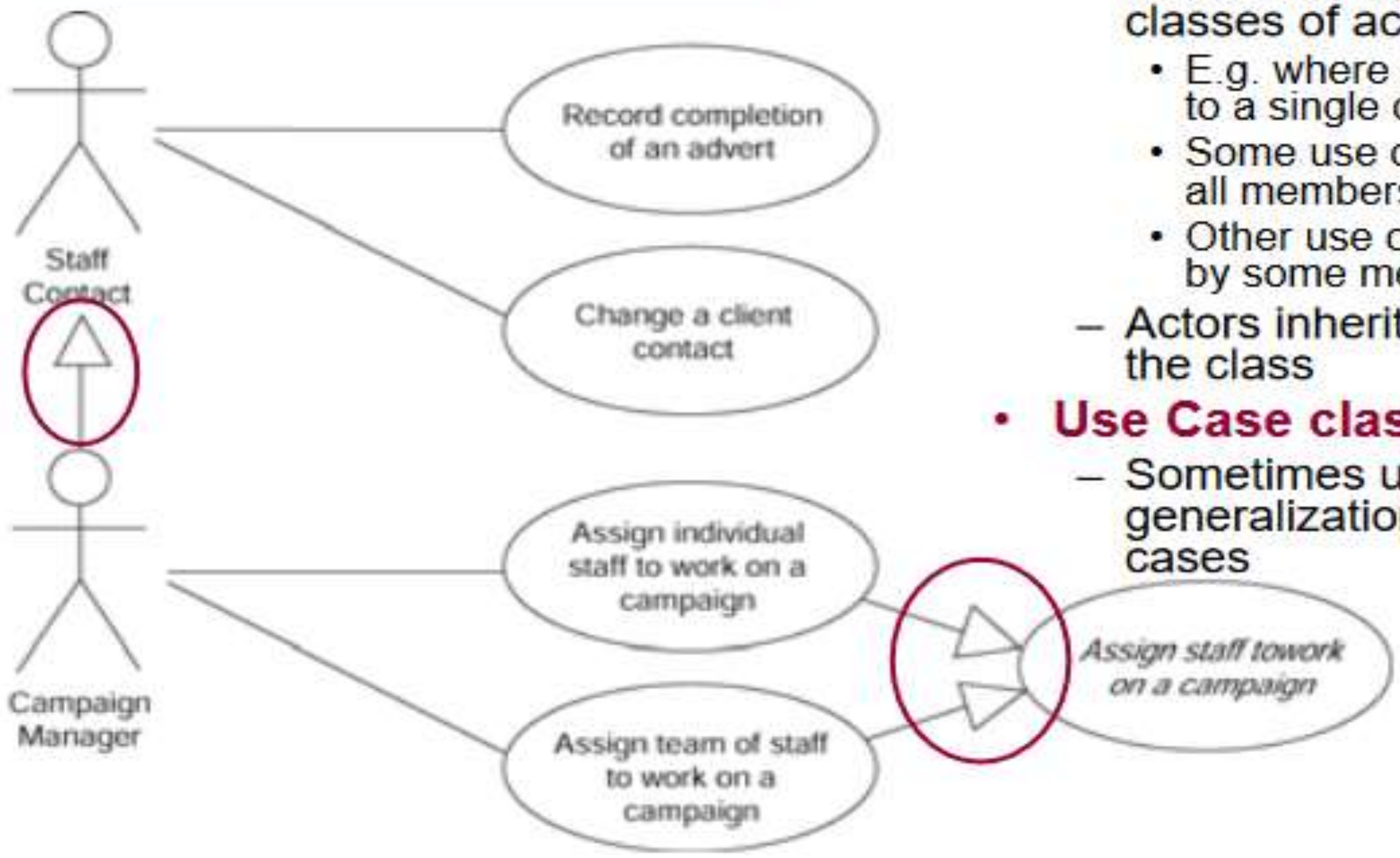
- inherit features from their parent use case;
- add new features;
- override (change) inherited features.

The child use case automatically inherits *all* features from its parent



Generalizations

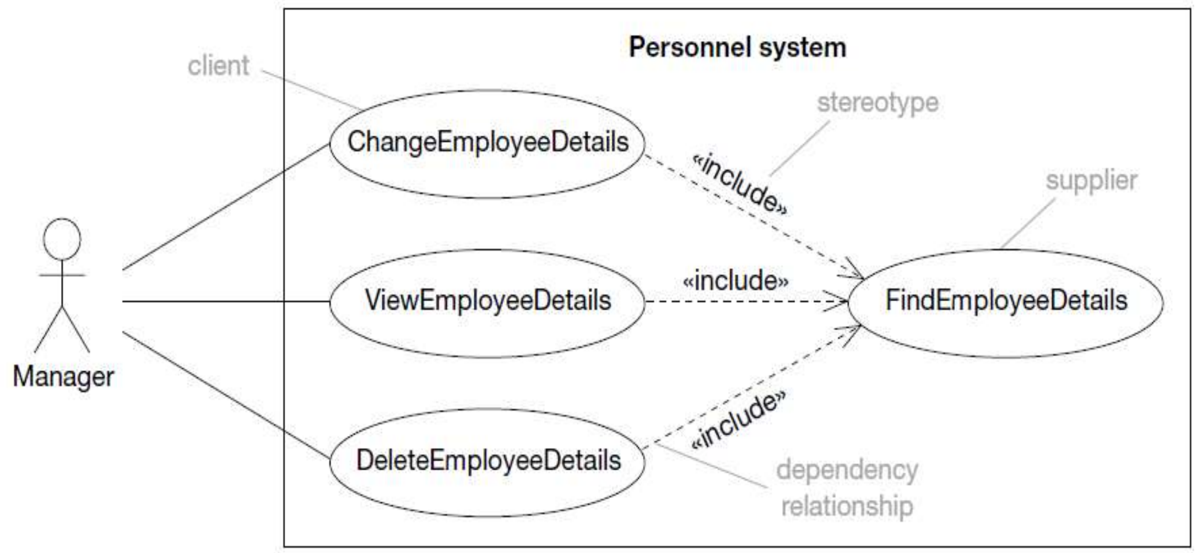
Generalization relations: "is a"


Record completion of an advert
Change a client contact
Staff Contact
Assign individual staff to work on a campaign
Assign team of staff to work on a campaign
Assign staff towork on a campaign
Campaign Manager

- **Actor classes**
  - It's sometimes useful to identify classes of actor
    - E.g. where several actors belong to a single class
    - Some use cases are needed by all members in the class
    - Other use cases are only needed by some members of the class
  - Actors inherit use cases from the class
- **Use Case classes**
  - Sometimes useful to identify a generalization of several use cases

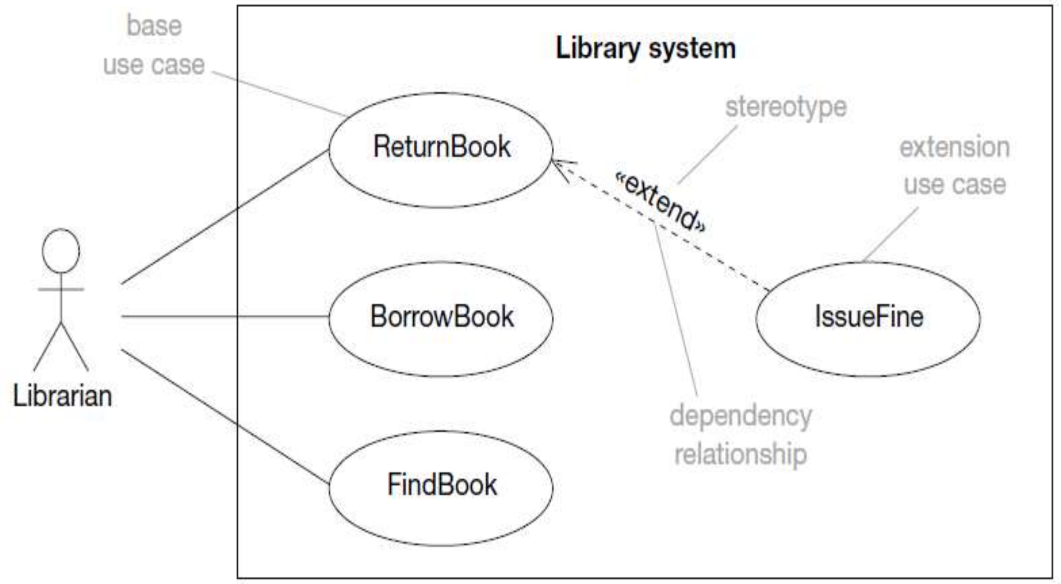
<<include>> relationships

Writing use cases can be very repetitive at times. Suppose you are writing a Personnel system. Almost anything we ask the system to do will first involve locating the details of a specific employee. If you had to write this sequence of events (involving user authentication, entering a user ID or some other unique identifier, etc.) every time you needed employee details, then your use cases would become quite repetitive. The «include» relationship between use cases allows you to include the behavior of a supplier use case into the flow of a client use case.



<<extend>> relationships

«extend» provides a way to add new behavior to an existing use case. The base use case provides a set of extension points which are hooks where new behavior may be added, and the extension use case provides a set of insertion segments that can be inserted into the base use case at these hooks. The «extend» relationship itself can, as you will see shortly, be used to specify *exactly* which extension points in the base use case are being extended.



Key references

UML and the Unified Process Practical object-oriented analysis and design-By-Jim Arlow , Neustadt

http://www.inf.ed.ac.uk/teaching/courses/seoc/2011_2012/notes/SEOC03_notes.pdf

http://www.cs.toronto.edu/~nn/csc340h/winter07/lectures/w10/L9-part1-6up.pdf

Thank You