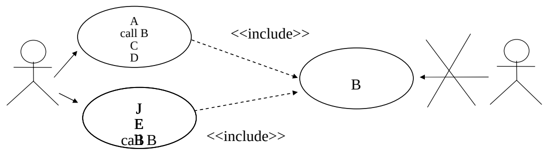
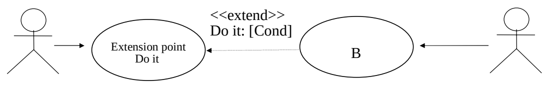
Use Cases - Include

- When two or more use cases share a part of behavior so the shared behavior is factorized in a new use case
- The new use case is not a normal use case, that is it will not be activated by an actor (90%)
- When the original use case is activated the included use cases are activated too (always)
- The include relationship is “explicit”



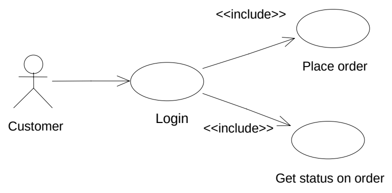
Use Cases - Extend

- If a condition is true a use case needs another to complete its behavior
- The new use case is a normal use case, that is it is activated by an actor (90%)
- When the original use case is activated according to the condition the extend use case is activated
- The extend relationship is “implicit”



Login Use Case

- Login use case that includes all of the other use cases that are secure
- In addition to updating the diagram, it is also necessary to write the text of the login use case. None of the other use cases change.



Login Use Case

1. The use case begins when the customer starts the application
2. The system prompts the customer to enter a username and password
3. The customer enter a username and password
4. The system verifies that this is a valid user
5. While the customer does not select exit, do the following steps in any order
6. The customer may choose to place an order (include Place Order)
7. The customer may choose to get the status on an order (include Get Status on Order)

end loop.

8. The use case ends.

Login Use Case

<u>Benefits</u>

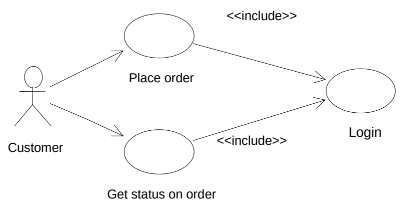
- The use case diagram looks like what the designer expect: The customer logs in, and form there he she can access any of the allowed system functions

<u>Drawback</u>

- The text of the use case is difficult to maintain in case a new use case is needed in the system. It is possible to forget to make the change to login
- It is not good that the login has the knowledge of all other parts of the system, Use cases have to be independent of each other

Login Use Case

- The other use cases include login
- It is necessary to update the use case diagram and the text



Login Use Case

Login Use Case

1. The use case begins when the customer starts the application

2. The system prompts the customer to enter a username and password

3. The customer enter a username and password

4. The system verifies that this is a valid user

5. The use case ends

Partial Place Order Use Case

1. The use case begins when the customer logs in to the system (include login)

Login Use Case

<u>Benefits</u>

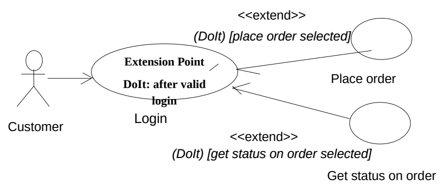
- **The login use case describes login and nothing else**

<u>Drawbacks</u>

- **The Place Order use case and the diagram make it appear that the customer has to log in every time s/he wants to do something**

Login Use Case

- **The other use cases extend the login use case**
- **It is necessary to update the use case diagram and the text**



Login Use Case

1. The use case begins when the customer starts the application

2. The system prompts the customer to enter a username and password

3. The customer enter a username and password

4. The system verifies that this is a valid user

5. While the customer does not select exit

6. Extension point: Do It

7. The use case ends

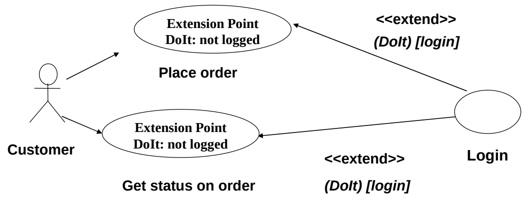
Login Use Case

Benefits

- **The customer logs one time and from there gets access to the resto of the system. Login adds an extension point, but that is all, it is no necessary to change it when adding new use cases**
- **Drawbacks**
- **The description of the extentio. It can be a difficult relationship to explain, especially to peole who are not developers**

Login Use Case

- **The login extend the other use cases**
- **It is necessary to update the use case diagram and the text**



Login Use Case

1. The use case begins when the customer starts the application
2. The system prompts the customer to enter a username and password
3. The customer enter a username and password
4. The system verifies that this is a valid user
5. While the customer does not select exit
6. The use case ends

Place Order Use Case

1. Extension point: Do It

Get Status on Order Use Case

1. Extension point: Do It

Login Use Case

Benefits

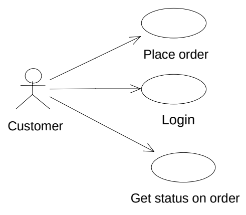
- The login only describes login and nothing else

Drawbacks

- The description of the extension relationship can be a difficult to explain, especially to peole who are not developers
- All the use cases has to adds the extension point

Login Use Case

- The login use case can be completely independent of the other use cases, but a precondition is included in the other use cases that a valid user be logged in before they can be executed
- It is necessary to update the use case diagram and the text



Login Use Case

1. The use case begins when the customer starts the application
2. The system prompts the customer to enter a username and password
3. The customer enter a username and password
4. The system verifies that this is a valid user
5. The use case ends

Partial Place Order Use Case

Assumption: A valid user has logged into the system

Login Use Case

Benefits

- The login only describes login and nothing else
- The diagram and the text are clear and easy to understand and the system is more flexible
- Place Order does not require login to execute, but just a valid user. Executing login is one way to get a valid user, but there may be other validation methods as well

Login Use Case

- The last approach seem to be the easiest to read and it has the most flexibility
- However, all the approaches are correct, so the developer can pick the one that works best for his/her project

Documenting Create, Read, Update, Delete

- This kind of use cases are often necessary in applications that include maintaining data
- Two approaches:
  - Create a separate use case for each kind of access to the data
  - Create one use case for a data type that embeds all of the CRUD functions
- When the use case is named by the behavior that the user expects, they are not called Create Order, Read Order, Update Order and Delete Order. The developer uses names that makes sense to the users of the system

Documenting Create, Read, Update, Delete

- It is possible to make one use case called something like Maintain Orders, which have been in charge of anything having to do with orders. This does not really make sense because different actors use separate CRUD functions, so it makes more sense to make them separate use cases
- On the other hand, in some applications combining them may be sensible, for example, if the application is for a database administrator to update tables in a database
- The best way is to begin with separate use cases and them analyze if the can be merged to make them easier to read and maintain