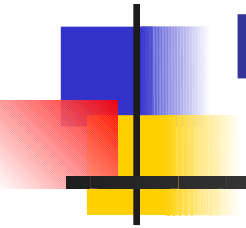
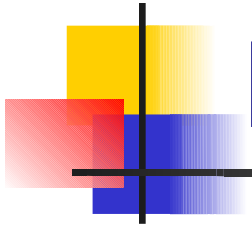
OO Analysis and Design with UML 2 and UP

Dr. Jim Arlow,
Zuhlke Engineering Limited

Requirements – advanced use case modelling



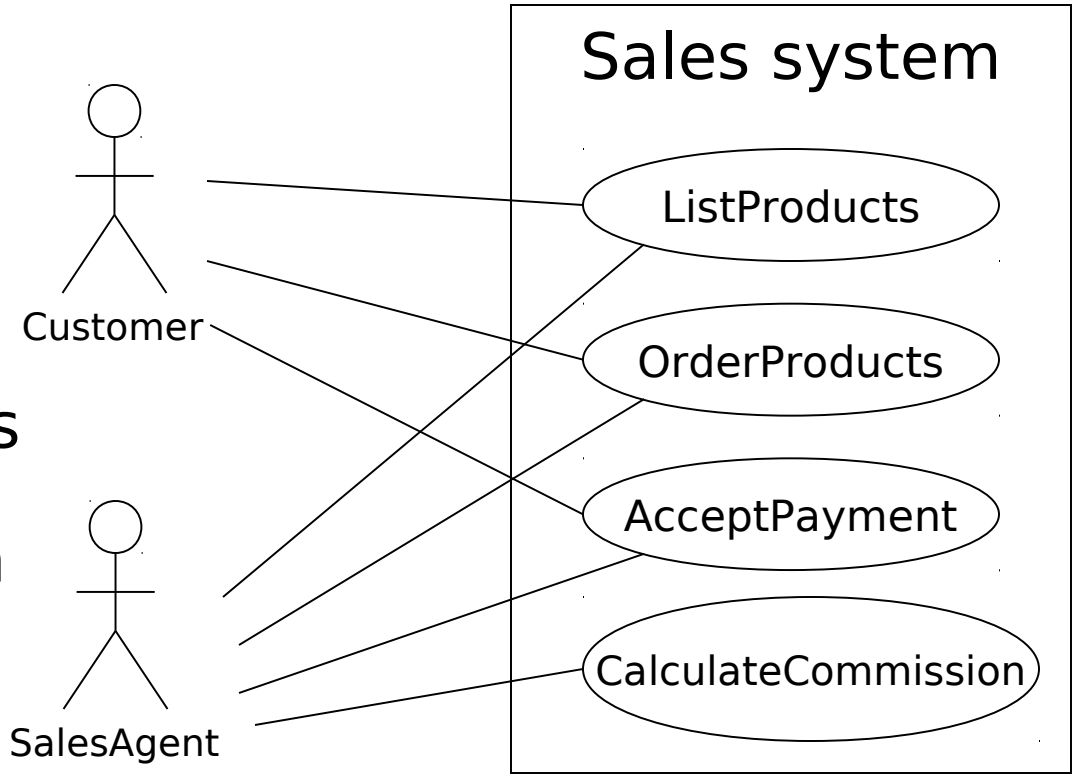


More relationships...

- We have studied basic use case analysis, but there are relationships that we have still to explore:
  - Actor generalisation
  - Use case generalisation
  - «include» – between use cases
  - «extend» – between use cases

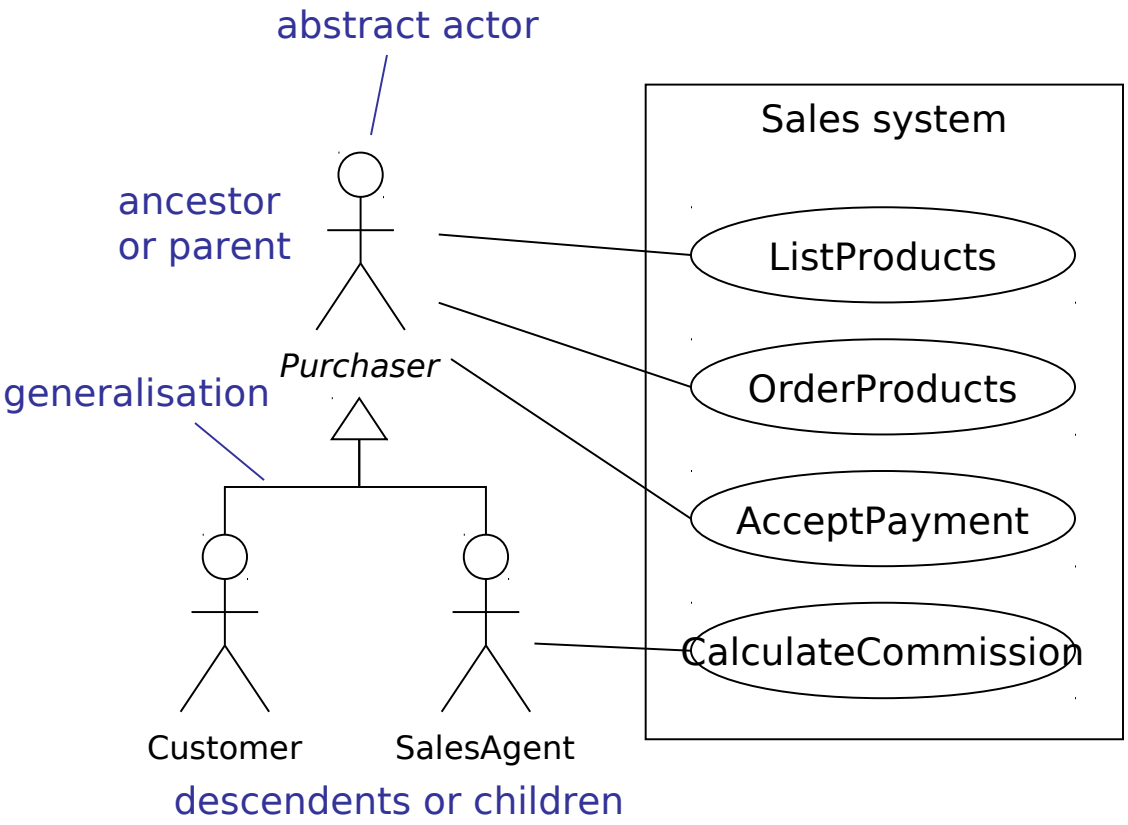
Actor generalization - example

- The Customer and the Sales Agent actors are *very* similar
- They both interact with List products, Order products, Accept payment
- Additionally, the Sales Agent interacts with Calculate commission
- Our diagram is a *mess* – can we simplify it?



Actor generalisation

- If two actors communicate with the same set of use cases in the same way, then we can express this as a generalisation to another (possibly abstract) actor
- The descendent actors inherit the roles and relationships to use cases held by the ancestor actor
- We can substitute a descendent actor anywhere the ancestor actor is expected. This is the *substitutability principle*

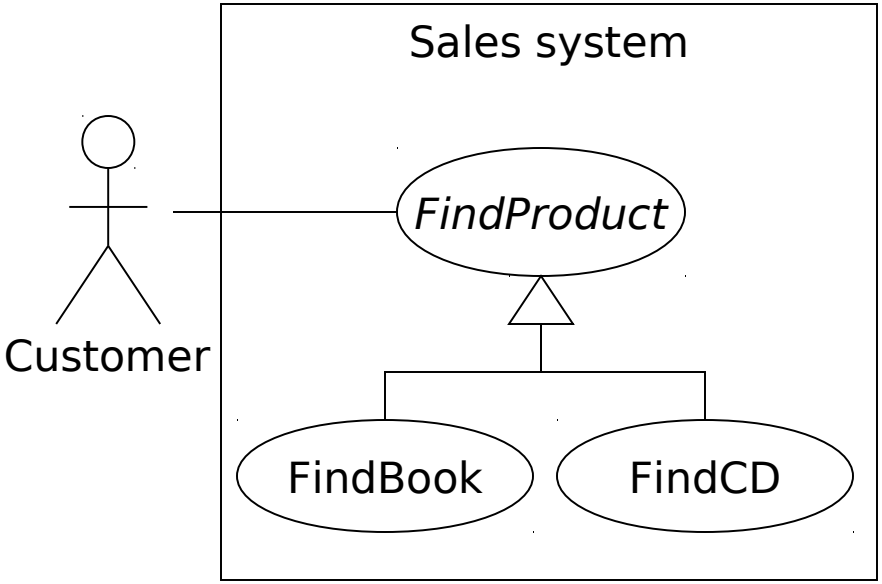


Use actor generalization when it simplifies the model

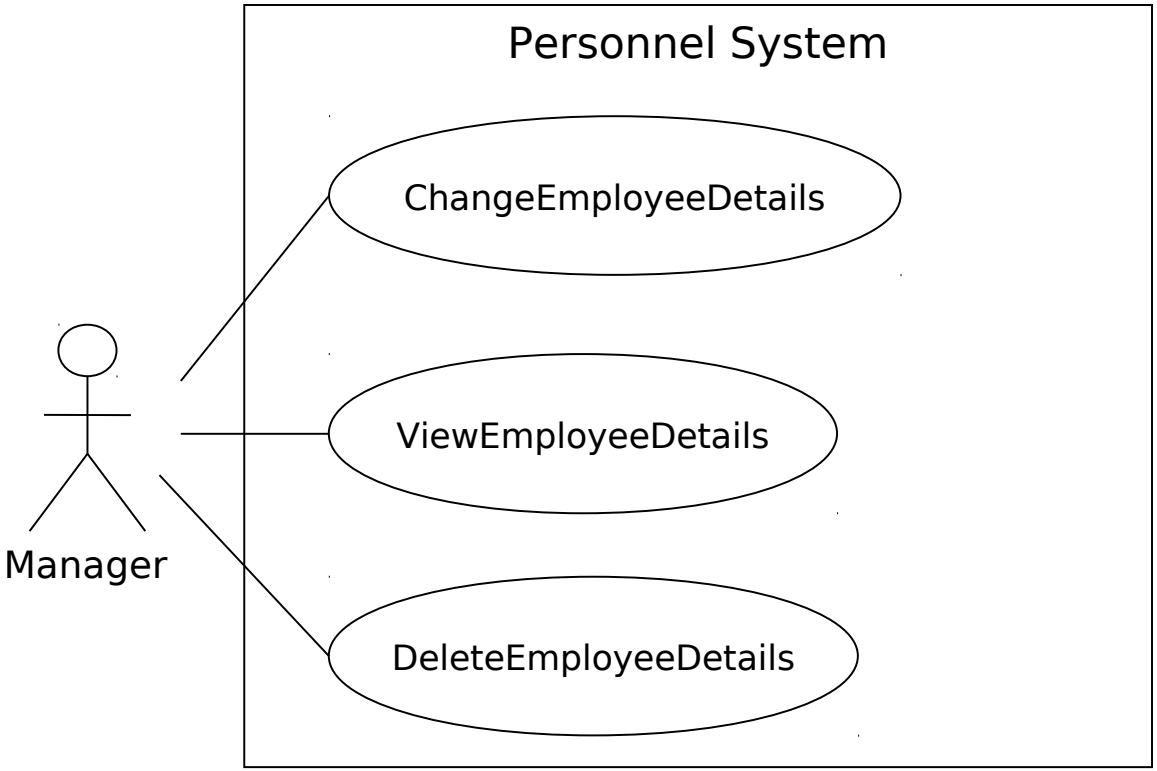
Use case generalisation

- The ancestor use case must be a more general case of one or more descendant use cases
- Child use cases are more specific forms of their parent
- They can inherit, add and override features of their parent

Use case generalization semantics			
Use case element	Inherit	Add	Override
Relationship	Yes	Yes	No
Extension point	Yes	Yes	No
Precondition	Yes	Yes	Yes
Postcondition	Yes	Yes	Yes
Step in main flow	Yes	Yes	Yes
Alternative flow	Yes	Yes	Yes

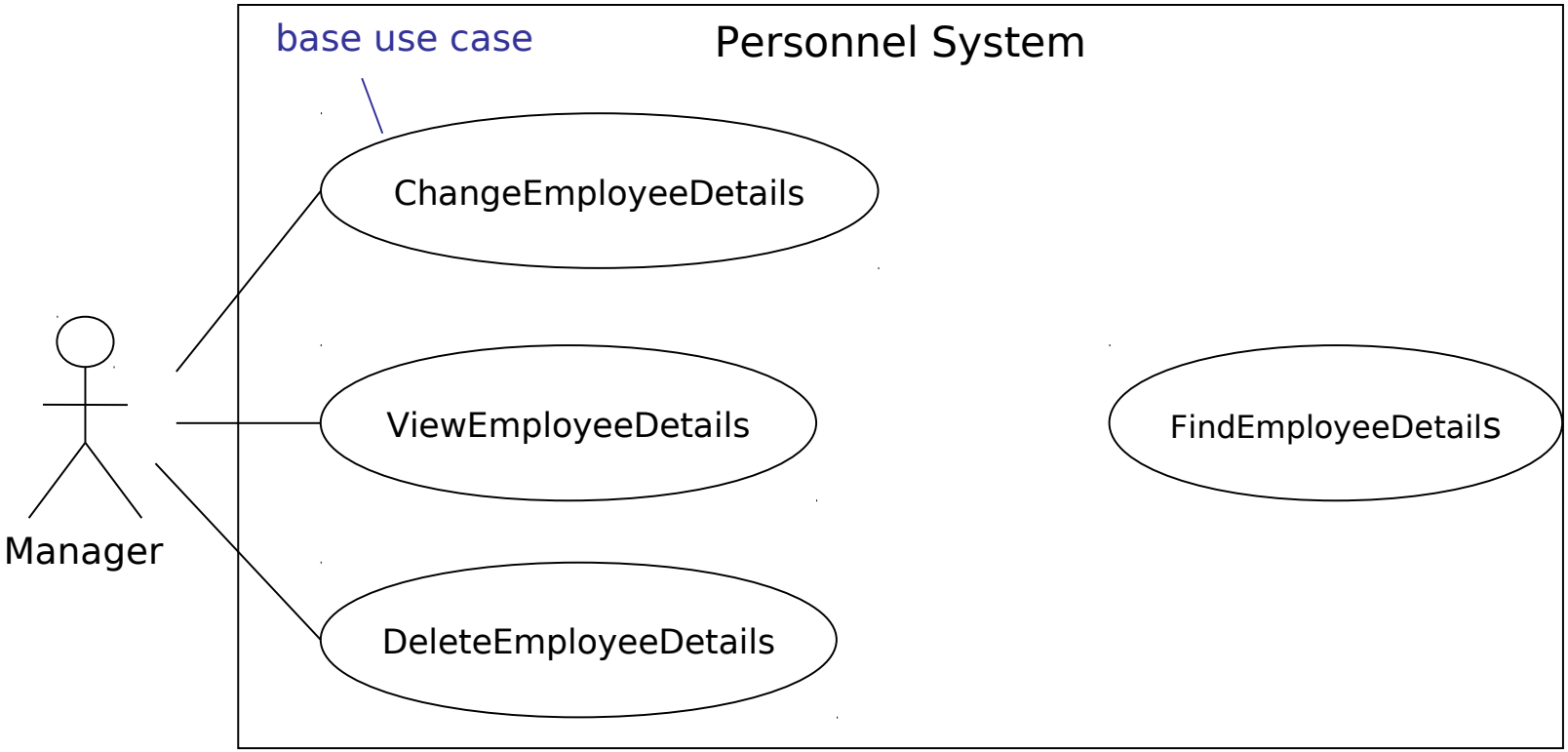


«include»



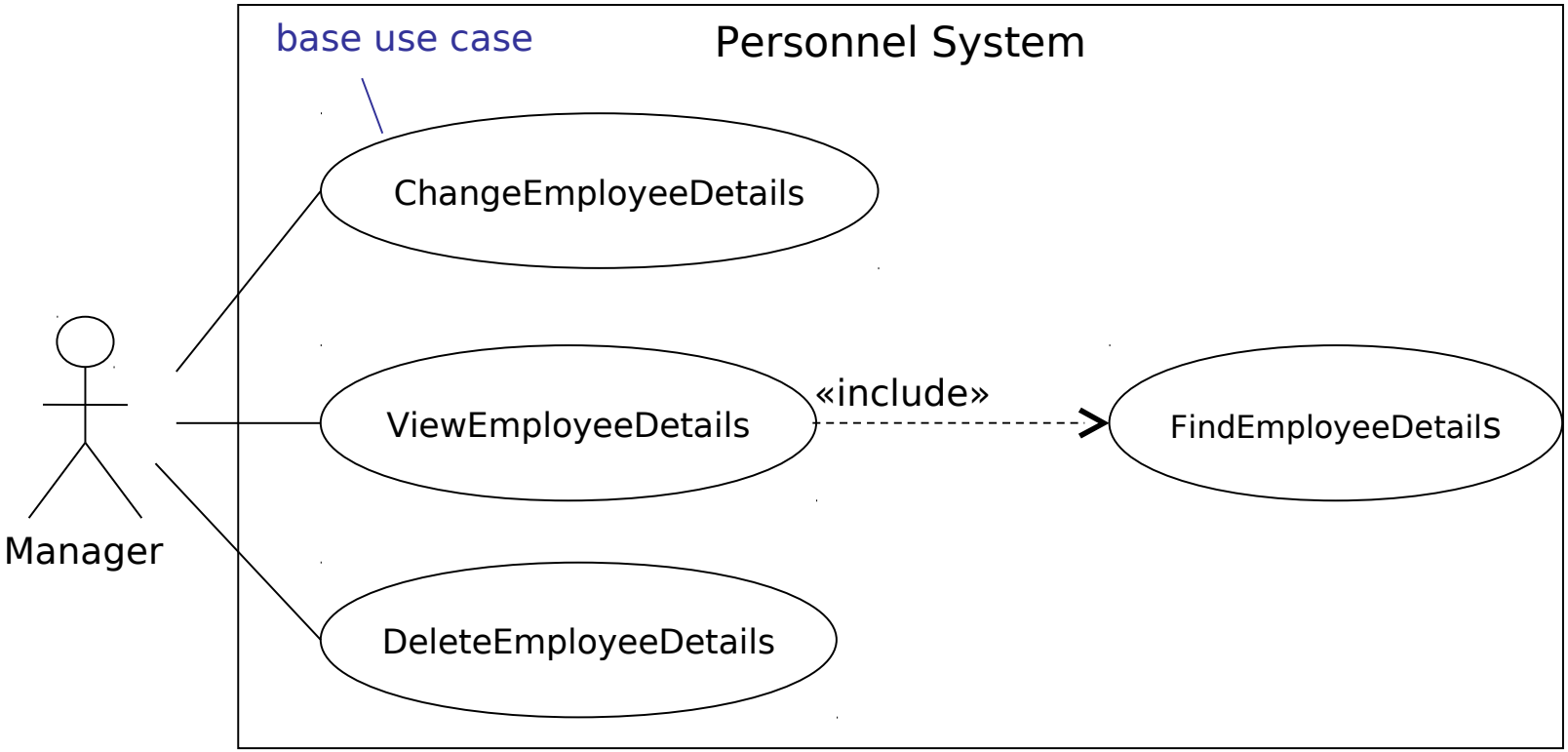
When use cases share common behaviour we can factor this out into a separate inclusion use case and «include» it in base use cases

«include»



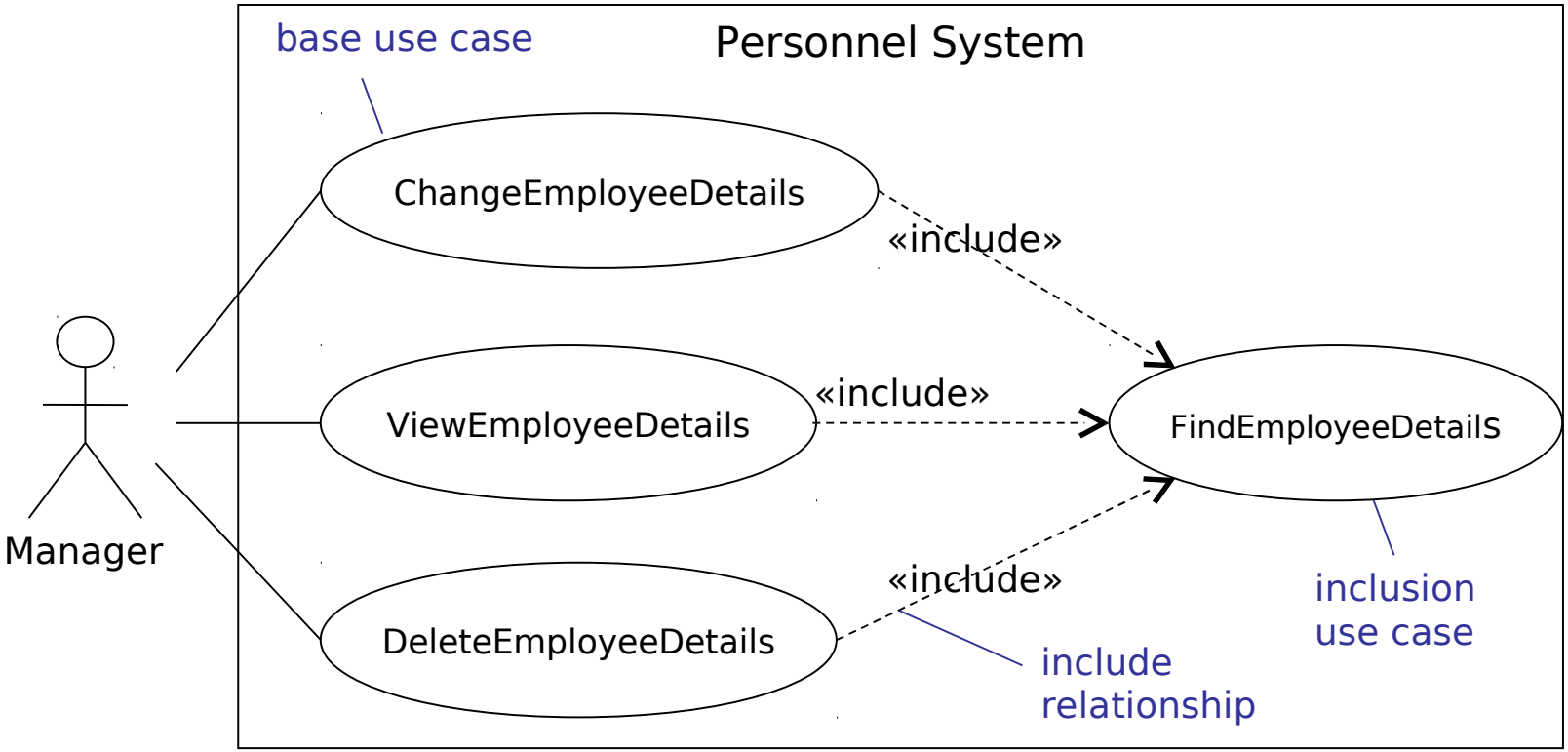
When use cases share common behaviour we can factor this out into a separate inclusion use case and «include» it in base use cases

«include»



When use cases share common behaviour we can factor this out into a separate inclusion use case and «include» it in base use cases

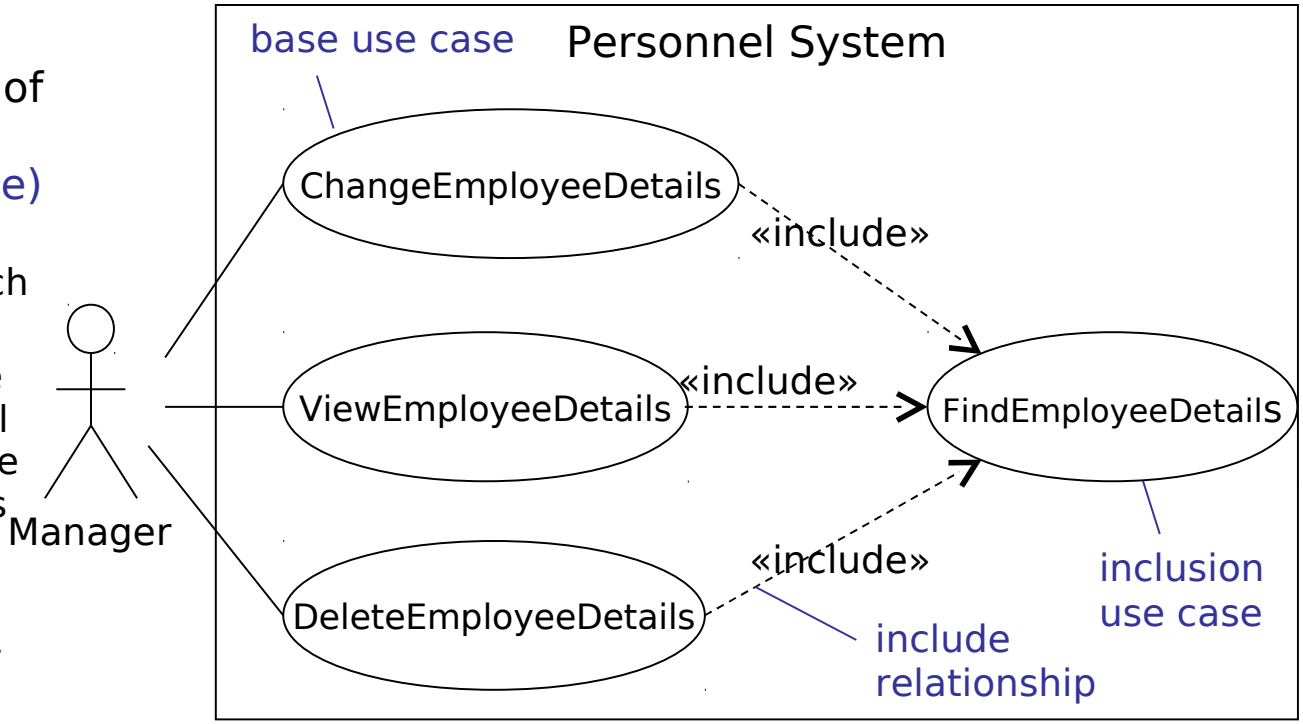
«include»



When use cases share common behaviour we can factor this out into a separate inclusion use case and «include» it in base use cases

«include»

- The base use case executes until the point of inclusion: include(InclusionUseCase)
  - Control passes to the inclusion use case which executes
  - When the inclusion use case is finished, control passes back to the base use case which finishes execution
- Note:
  - Base use cases are *not complete* without the included use cases
  - Inclusion use cases may be complete use cases, or they may just specify a fragment of behaviour for inclusion elsewhere



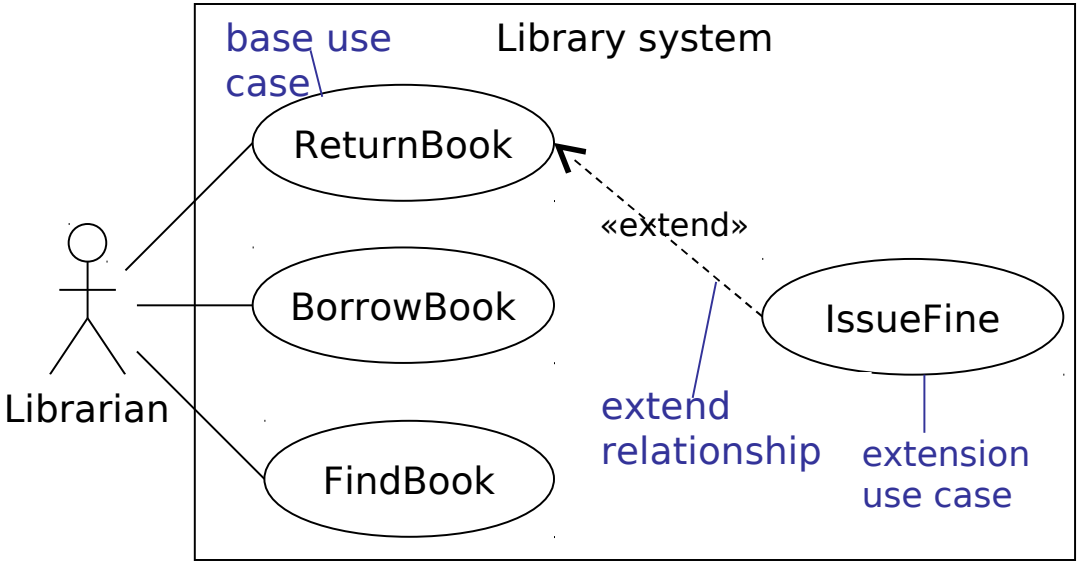
«include» example

Use case: ChangeEmployeeDetails
ID: 1
Brief description: The Manager changes the employee details.
Primary actors: Manager
Seconday actors: None
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. include(FindEmployeeDetails). 2. The system displays the employee details. 3. The Manager changes the employee details.
Postconditions: 1. The employee details have been changed.
Alternative flows: None.

Use case: FindEmployeeDetails
ID: 4
Brief description: The Manager finds the employee details.
Primary actors: Manager
Seconday actors: None
Preconditions: 1. The Manager is logged on to the system.
Main flow: 1. The Manager enters the employee's ID. 2. The system finds the employee details.
Postconditions: 1. The system has found the employee details.
Alternative flows: None.

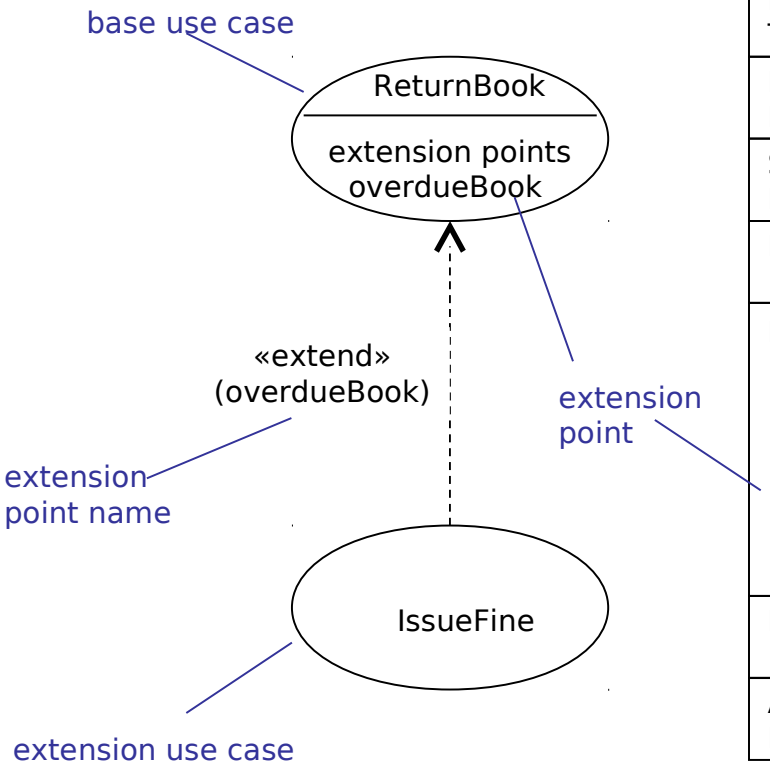
«extend»

- «extend» is a way of adding new behaviour into the base use case by inserting behaviour from one or more extension use cases
  - The base use case specifies one or more extension points in its flow of events
- The extension use case may contain several insertion segments
- The «extend» relationship may specify *which* of the base use case extension points it is extending



The extension use case inserts behaviour into the base use case.
The base use case provides extension points, but *does not know* about the extensions.

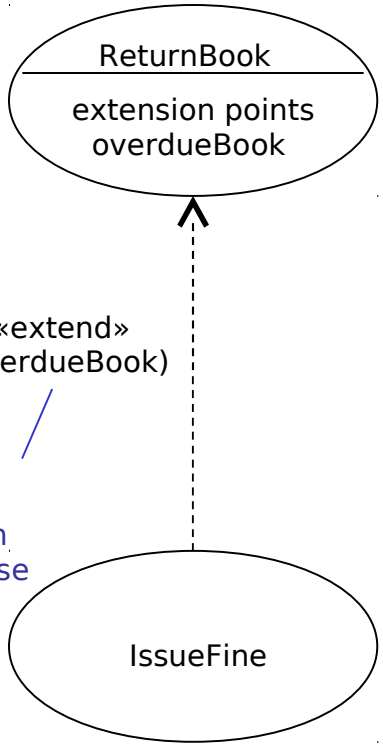
Base use case



Use case: ReturnBook
ID: 9
Brief description: The Librarian returns a borrowed book.
Primary actors: Librarian
Secondary actors: None.
Preconditions: 1. The Librarian is logged on to the system.
Main flow: 1. The Librarian enters the borrower's ID number. 2. The system displays the borrower's details including the list of borrowed books. 3. The Librarian finds the book to be returned in the list of books. extension point: overdueBook 4. The Librarian returns the book.
Post-conditions: 1. The book has been returned.
Alternative flows: None.

- There is an extension point overdueBook just before step 4 of the flow of events
- Extension points are *not* numbered, as they are *not* part of the flow

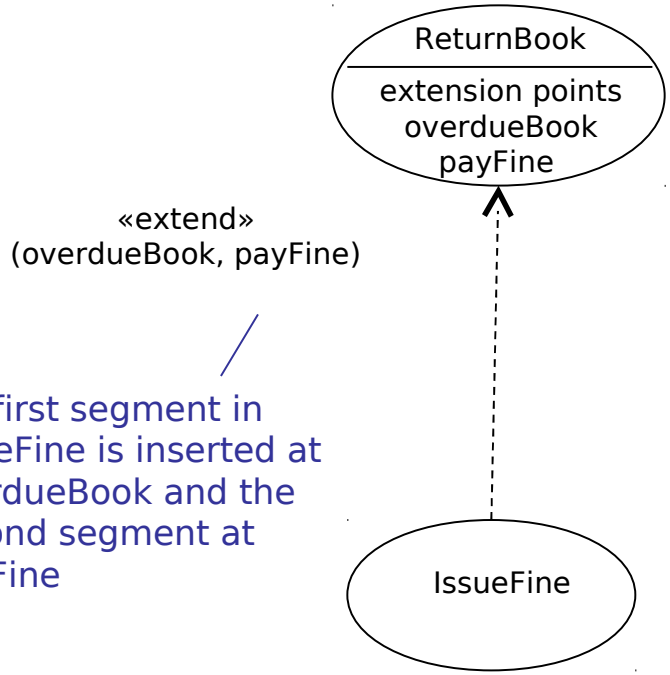
Extension use case



Extension Use case: IssueFine
ID: 10
Brief description: Segment 1: The Librarian records and prints out a fine.
Primary actors: Librarian
Secondary actors: None.
Segment 1 preconditions: 1. The returned book is overdue.
Segment 1 flow: 1. The Librarian enters details of the fine into the system. 2. The system prints out the fine.
Segment 1 postconditions: 1. The fine has been recorded in the system. 2. The system has printed out the fine.

- Extension use cases have one or more *insertion segments* which are behaviour fragments that will be inserted at the specified extension points in the base use case

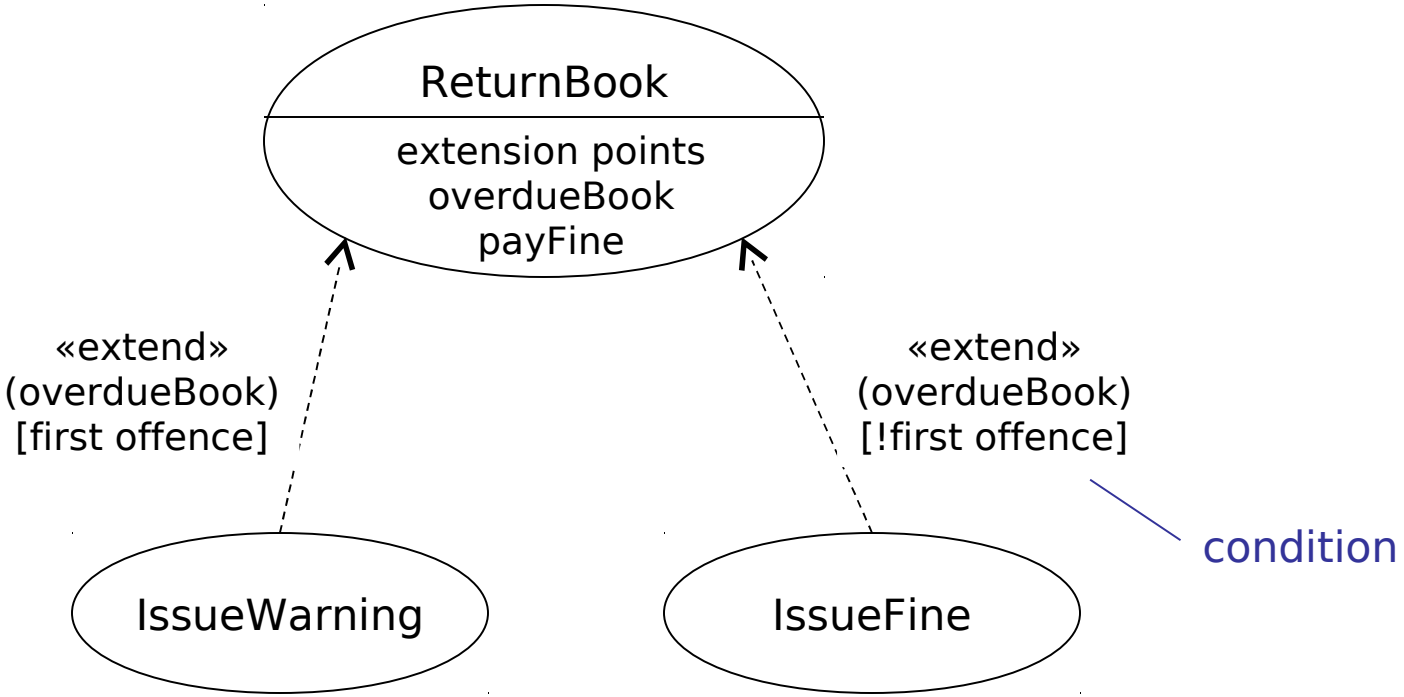
Multiple insertion points



- If more than one extension point is specified in the «extend» relationship then the extension use case must have the *same number* of insertion segments.

Extension Use case: IssueFine
ID: 10
Brief description: Segment 1: The Librarian records and prints out a fine. Segment 2: The Librarian accepts payment for a fine.
Primary actors: Librarian
Secondary actors: None.
Segment 1 preconditions: 1. The returned book is overdue.
Segment 1 flow: 1. The Librarian enters details of the fine into the system. 2. The system prints out the fine.
Segment 1 postconditions: 1. The fine has been recorded in the system. 2. The system has printed out the fine.
Segment 2 preconditions: 1. A fine is due from the borrower.
Segment 2 flow: 1. The Librarian accepts payment for the fine from the borrower. 2. The Librarian enters the paid fine in the system. 3. The system prints out a receipt for the paid fine.
Segment 2 postconditions: 1. The fine is recorded as paid. 2. The system has printed a receipt for the fine.

Conditional extensions



- We can specify conditions on «extend» relationships
  - Conditions are Boolean expressions
  - The insertion is made if and only if the condition evaluates to true

Rady a tipy pro psaní UC

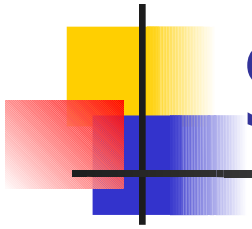
- Tvořit co nejkratší a nejjednodušší UC
- Soustředit se na CO, nikoli JAK
  - 4. Systém požádá Zákazníka o potvrzení objednávky.
  - 5. Zákazník stiskne tlačítko OK.

Rady a tipy pro psaní UC

- Tvořit co nejkratší a nejjednodušší UC
- Soustředit se na co, nikoli jak
  - 4. Systém požádá Zákazníka o potvrzení objednávky.
  - ~~5. Zákazník stiskne tlačítko OK.~~

Rady a tipy pro psaní UC

- Tvořit co nejkratší a nejjednodušší UC
- Soustředit se na co, nikoli jak
  - 4. Systém požádá Zákazníka o potvrzení objednávky.
  - ~~5. Zákazník stiskne tlačítko OK.~~
  - 5. Zákazník odsouhlasí objednávku.
- Vyhýbat se funkční dekompozici



Summary

- We have learned about techniques for advanced use case modelling:
  - Actor generalisation
  - Use case generalisation
  - «include»
  - «extend»
- Use advanced features with discretion only where they simplify the model!