WHITESTEIN Technologies

Unified Modeling Language

Use Cases

Radovan Cervenka

Use Case Model

→ **Functionality of the system and its surroundings.**

Consists of:

- Use Case diagrams
- Use Case and Actor descriptions

Supported by:

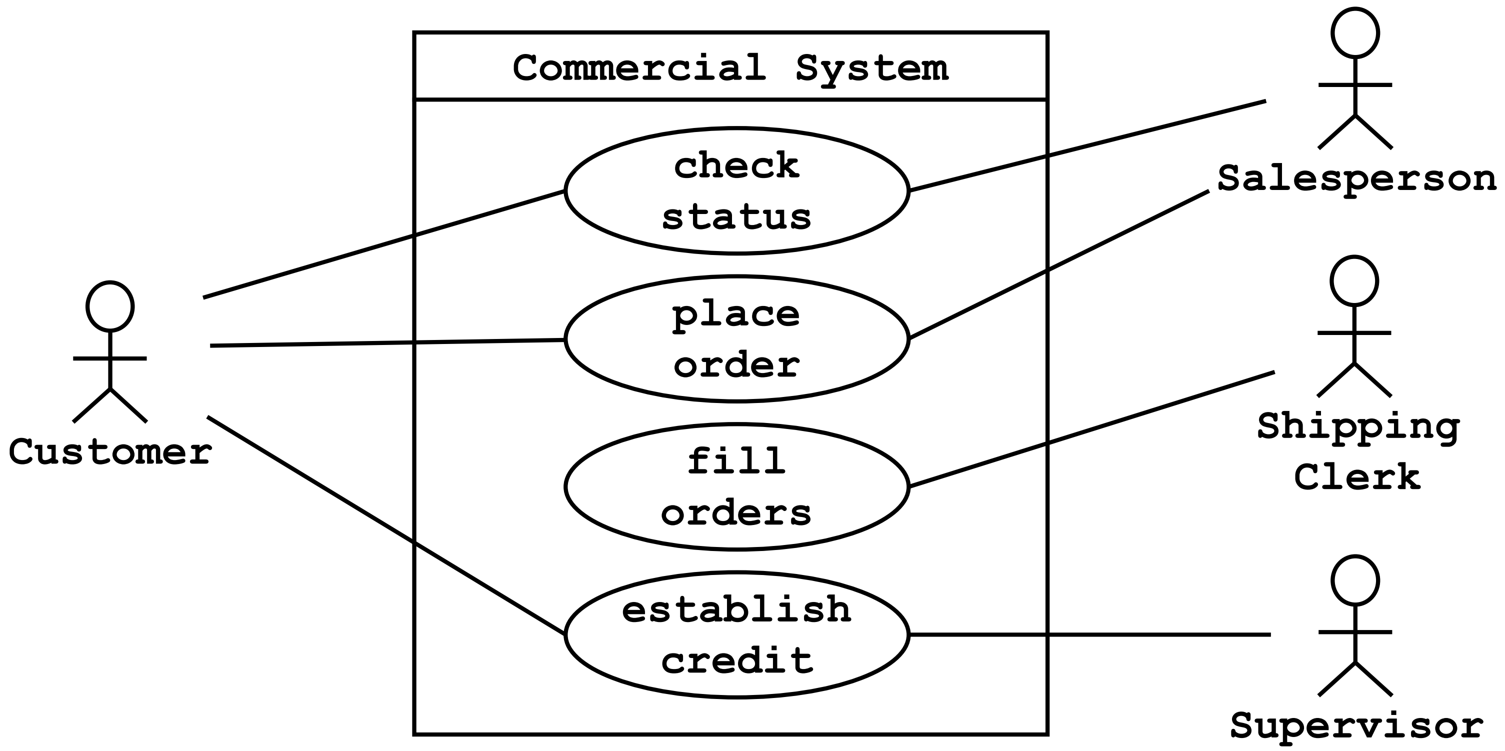
- UI descriptions (incl. prototypes)
- Specification of non-functional requirements

Used (mainly) in:

- Requirements $\Rightarrow$ functional system requirements
- Analysis and design $\Rightarrow$ analysis and design models
- Testing $\Rightarrow$ test cases
- Management $\Rightarrow$ planning and tracking

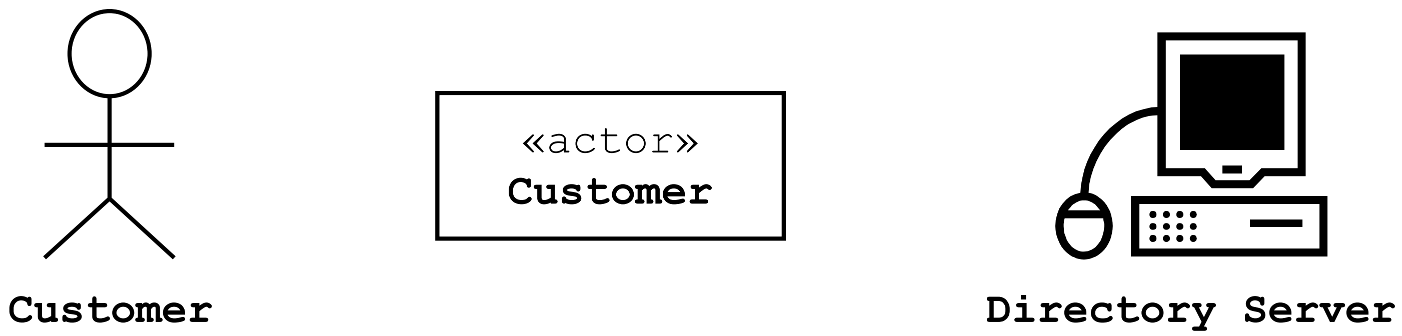
Use Case Diagram

- Defines outer behavior/functionality/required usages of a system.
- *Actors*, *use cases*, *subjects* and their relationships.



Actor

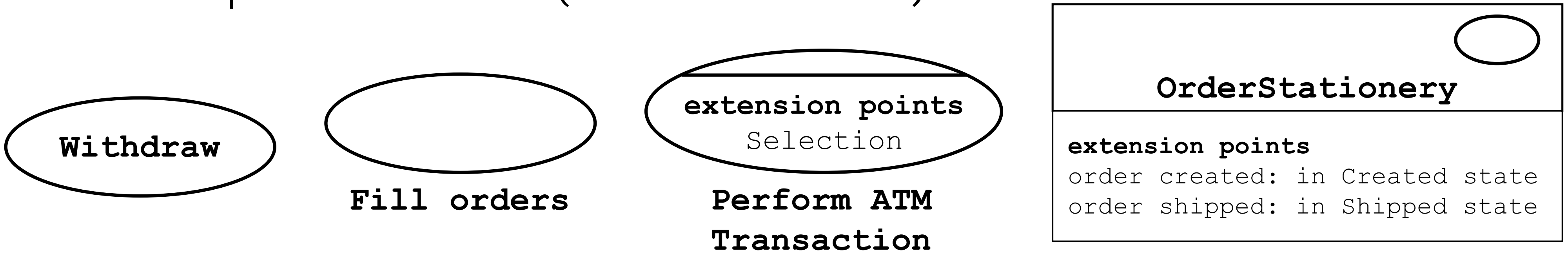
- → A role played by an entity that interacts with the subject (system).
- ■ External to the subject.
- ■ Represent roles played by human users, external hardware, or other subjects.
- ■ A single physical instance may play the role of several different actors
- ■ A given actor may be played by multiple different instances.
- ■ A specialized classifier (behaviored classifier).



Use Case

→ The specification of a set of actions performed by a system, which yields an observable result that is, typically, of value for one or more actors or other stakeholders of the system.

- A coherent behavior, possibly including variants (exception of error handling), that the subject can perform in collaboration with one or more actors.
  - Primary actor initiates the use case.
  - Secondary actor is used to complete the use case.
- Can be internally described by interactions, activities, and state machines, or by pre-conditions and post-conditions as well as by natural language text where appropriate.
- A specialized classifier (behaviored classifier).



Scenario

- A session that an actor instance has with the system.
- Has details of real data and actual expected output.
- Potentially hundreds to thousands in an application.

1
John enters his account# 404504
John enters his pin# 9342
John requests his average balance from 1/1/97 - 7/31/97
System gives the average balance

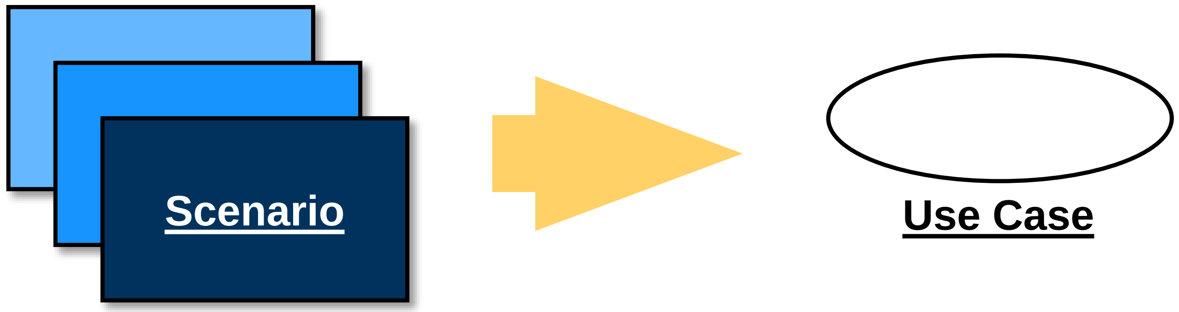
2
Larry enters his account# 4343443
Larry enters his pin# 84954
Larry requests his average balance from 1/1/95 - 12/31/96
System gives the average balance

3
Mary enters her account# 34334
Mary enters her pin# 4343
System detects non valid pin and repeats the procedure

Use Case vs. Scenario

Use cases are not scenarios!!!

- Use case represents a set of potential scenarios.
- Looking at a family of similar scenarios, you can gather the essence of what is typically done.
- Similar scenarios will follow similar patterns of work and provide similar types of results.
- Normally each use case focuses on a specific goal.
  - E.g. to obtain the current account balance.



Use Case Description

CHARACTERISTIC

- **Name:** use case name
- **Goal:** a longer statement of the goal
- **Scope:** subsystem, application, …
- **Pre-conditions:** state before use case execution
- **Post-conditions:** state after use case execution
- **Trigger:** action upon which is use case started
- **Primary actor:** a role name
- **Secondary actors:** list of other needed roles or systems

ALGHORITHM
(primary scenario, extensions and variations)

- steps of the algorithm:
  - <step#> <action description>
- control expressions:
  - if then else, repeat, switch, …

RELATED INFORMATION

- **Priority:** how critical
- **Time:** a performance duration
- **Frequency:** how often
- **Supplementary specifications:** other non-functional requirements

Main success scenario, extensions and variations (Cockburn)

- This is an alternative way how to describe use case algorithm.
- The algorithm is divided to 3 parts:
  - *Main success scenario*
    - A sequence of steps (usually linear) leading to a successful execution of the use case in which nothing goes wrong.
  - *Extensions*
    - Description of handling branches of the main success scenario. Each extension refers to a step of the main success scenario which is being extended, describes a condition under which it happens, and lists a number of steps used to handle the extension. Extensions are used to describe both, success and failure (exceptional) execution.
  - *(Technology and data) variations*
    - Description of different ways how the steps of the main success scenario can be executed; ie., different ways of execution or different types of data can be used.

Example of main success scenario, extensions and variations

- Use Case: Buy Goods
- MAIN SUCCESS SCENARIO:
  1. Buyer calls in with a purchase request.
  2. Company captures buyer’s name, address, requested goods, etc.
  3. Company gives buyer information on goods, prices, delivery dates, etc.
  4. Buyer signs for order.
  5. Company creates order, ships order to buyer.
  6. Company ships invoice to buyer.
  7. Buyers pays invoice.
- EXTENSIONS:

  3a. Company is out of one of the ordered items:

  3a1. Renegotiate order.

  4a. Buyer pays directly with credit card:

  4a1. Take payment by credit card

  7a. Buyer returns goods:

  7a1. Handle returned goods
- VARIATIONS:

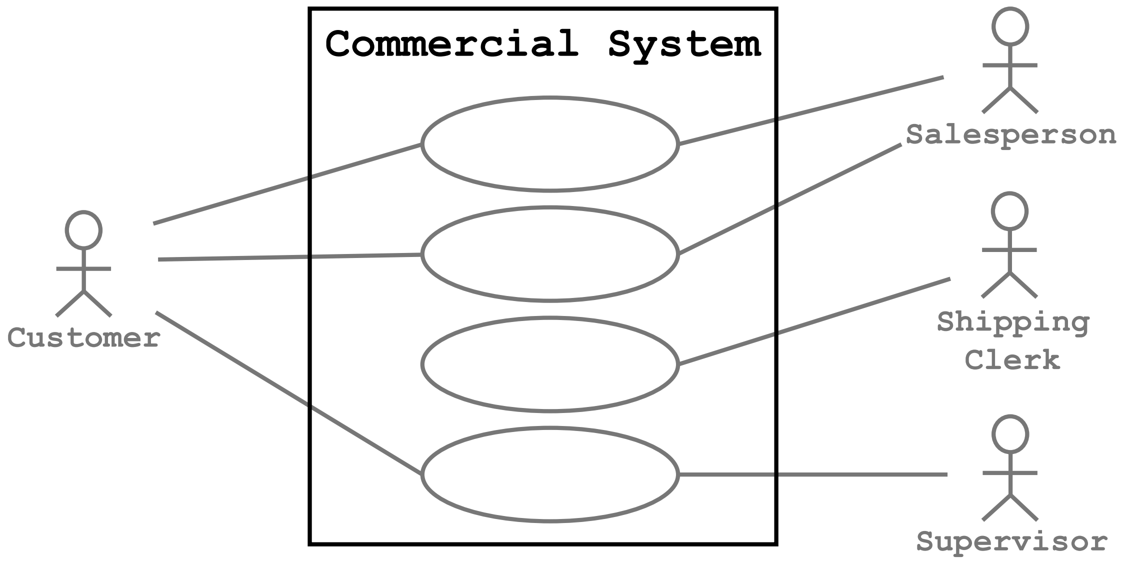
  1’. Buyer may use phone, fax, or web order form

  2’. With or without company info.

  2’’. With or without discount request.

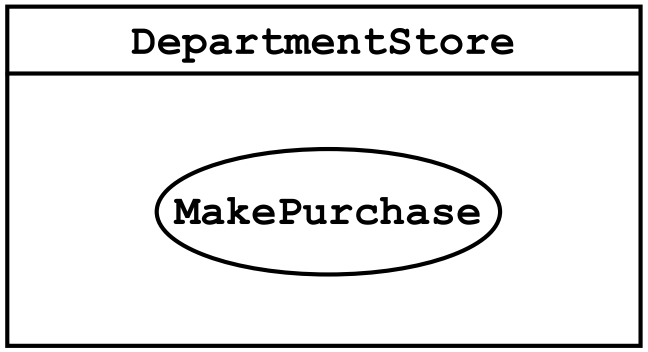
System (UML 1.*)

- → The functional boundary of the system.
- ■ Described by a finite set of use cases.
- ■ Actors are drawn outside of the system, use cases inside.



Classifier as a Subject (UML 2.*)

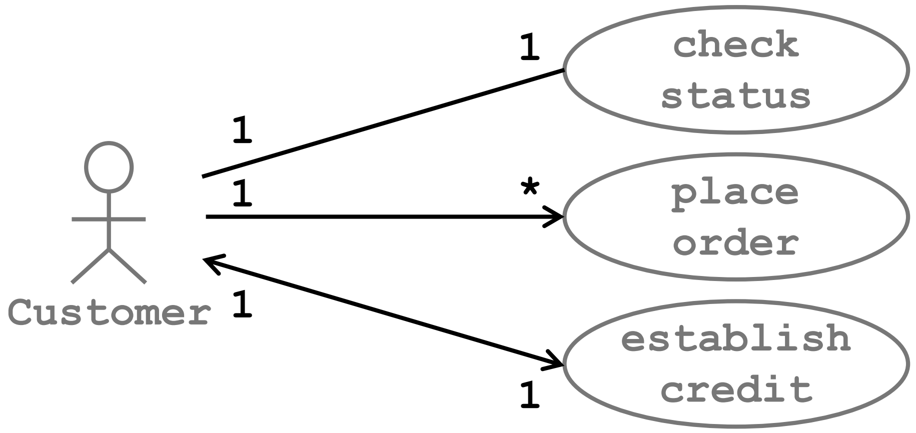
- A classifier with the capability to own use cases.
- Typically represents the subject to which the owned use cases apply.



Association (in Use Case Diagrams)

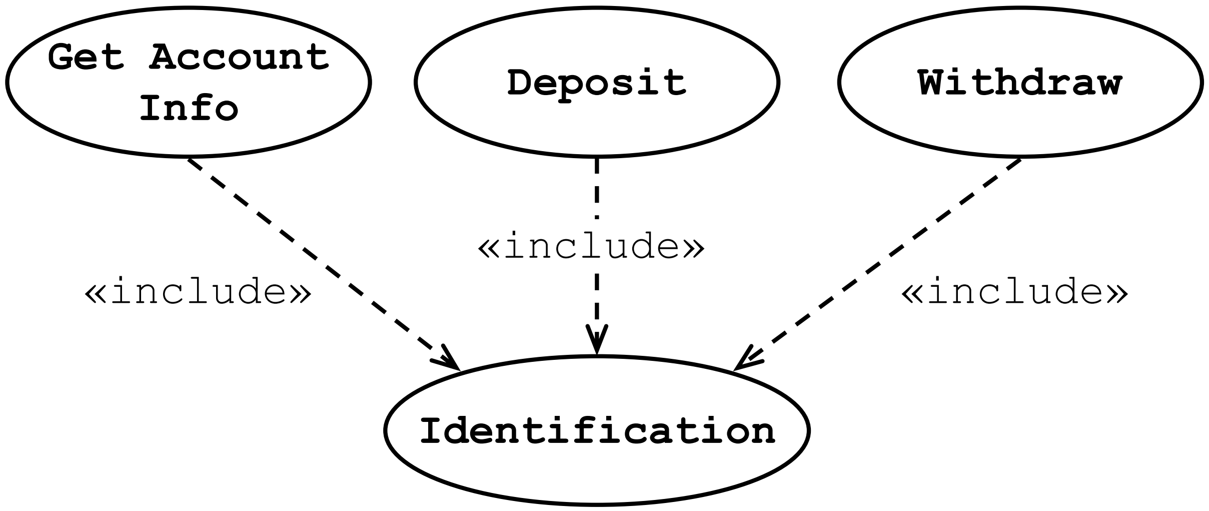
- A (binary) relationship between an actor and a use case meaning interaction of the actor with the use case's subject.
- Multiplicity
  - The range of allowable instances at the association end.
  - A list of values and intervals expressed as: lower limit .. upper limit
  - Possible values : number or * (many)

    '1', '*', '0..*', '1..*', '2, 4, 6..10, 20..*'
- Navigability
  - Indicates that navigation is supported toward attached classifier.



Include

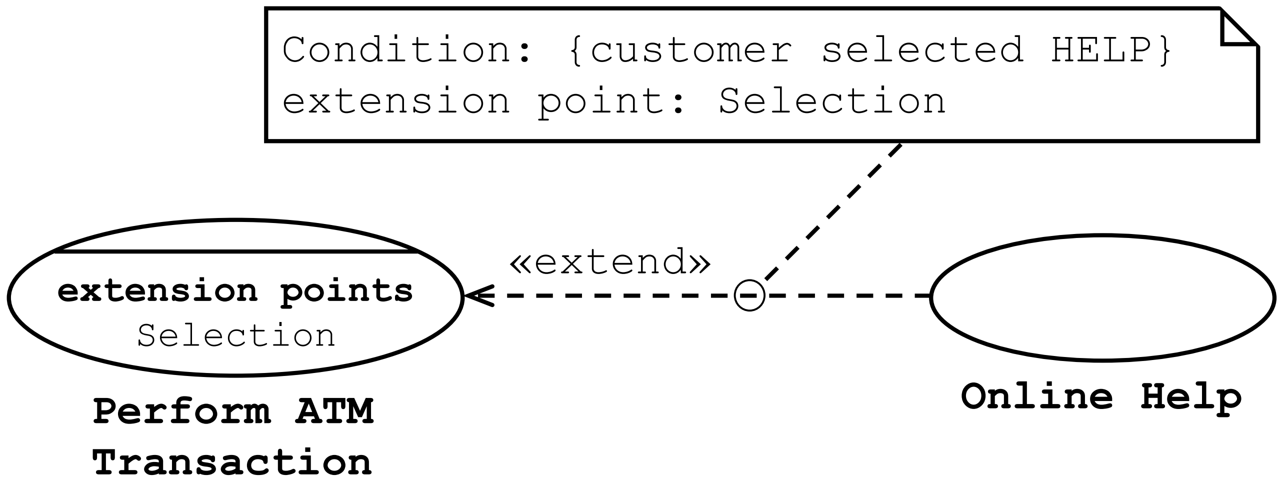
- → Indicates that an *including* use case contains the behavior defined in another, *included* (base), use case.
- ■ Included use case obviously implements a behavior shared by a number of use cases (“subroutine”).
- ■ The included use case is always required for the including use case to execute correctly.



Extend

→ A relationship from an *extending* use case to an *extended* use case that specifies how and when the behavior defined in the extending use case can be inserted into the behavior defined in the extended use case.

- The extension takes place at one or more specific extension points defined in the extended use case.
- The extended use case is defined independently of the extending use case.
- The extending use case usually depends on the extended use case.
- Extension often represents unusual behavior, such as an exception or error handling.



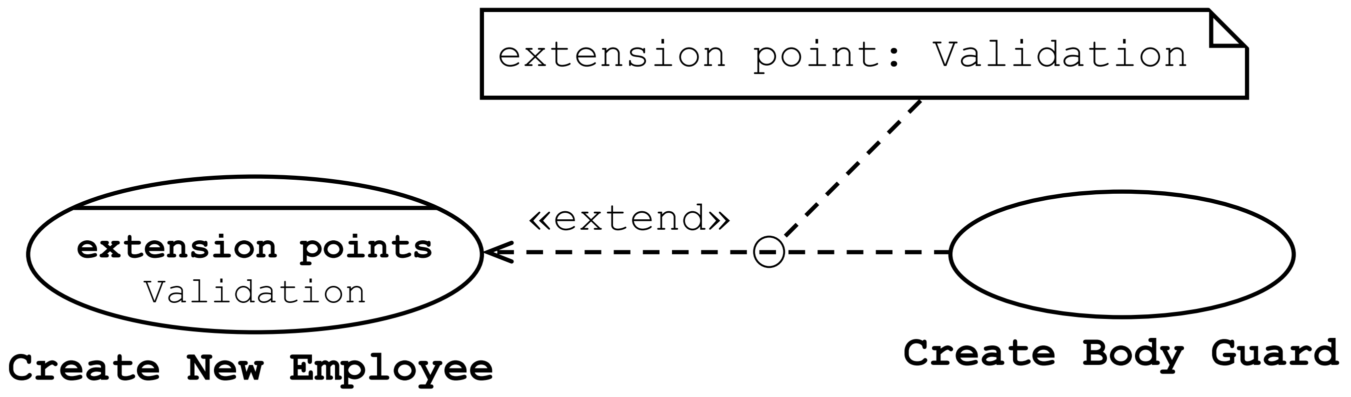
Example of Extension

Create New Employee - extended use case

1. Enter employee information
2. Enter salary information
3. Enter job title
4. Save entry
5. **Validation:** System validates
6. If no problems, systems setups new employee record

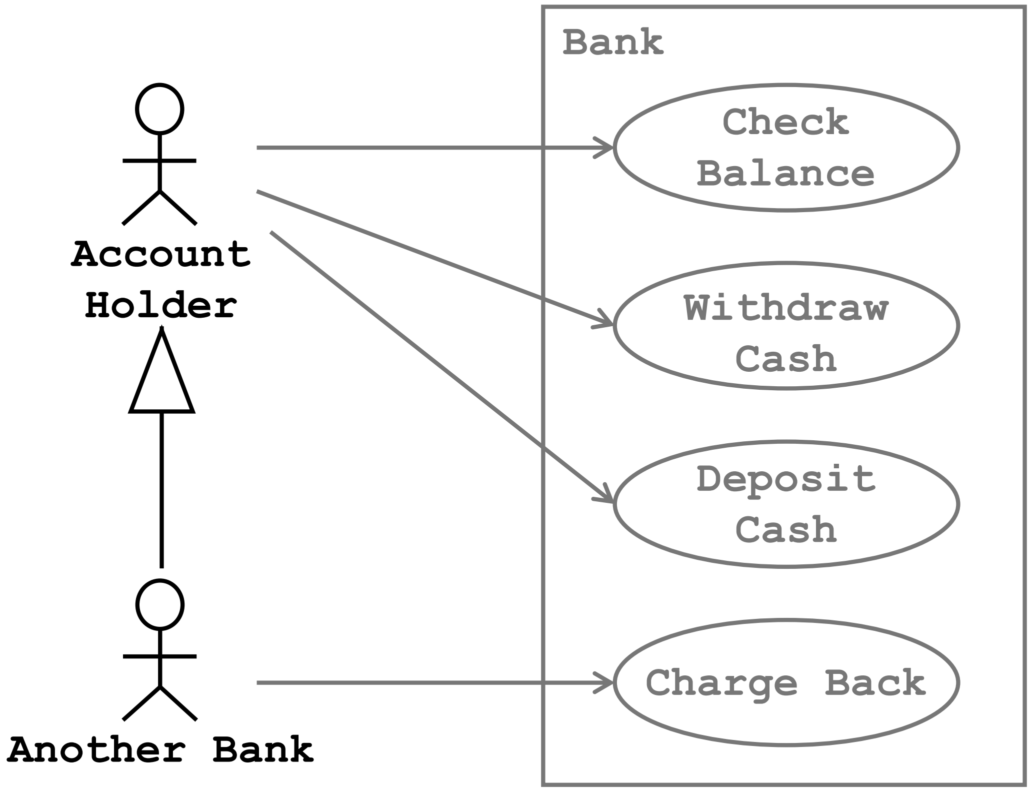
Create Body Guard - extending use case

- System checks employee police record



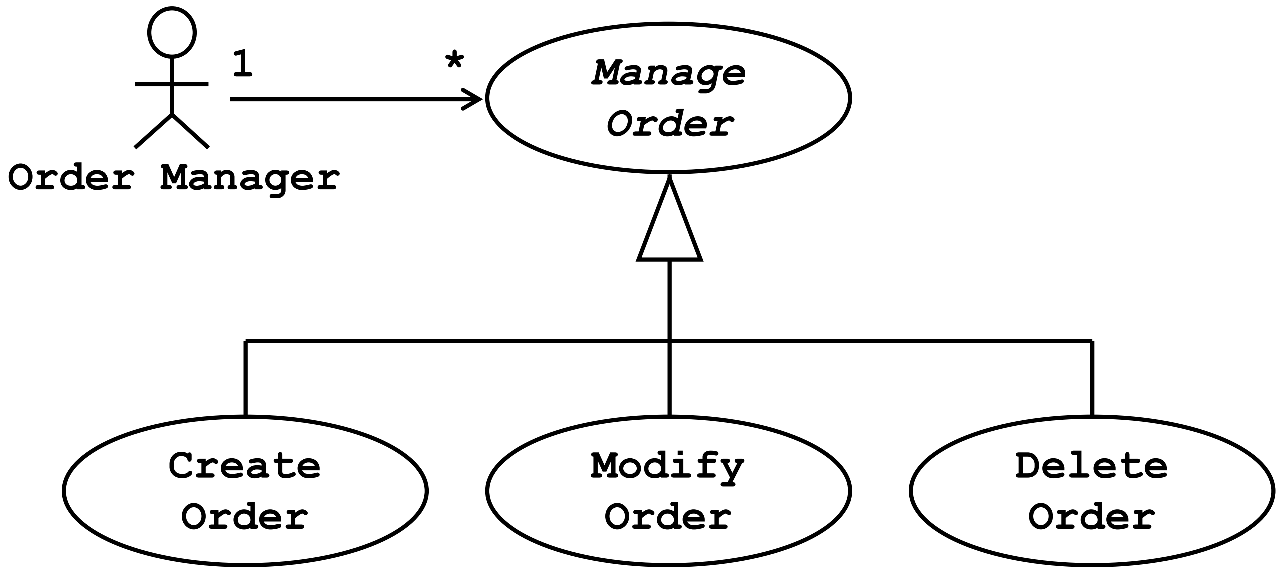
Generalization of Actors

- The specific (child) actor gets all capabilities of the general (parent) actor.
- Simplifies the diagram, without losing semantics.
- Abstract actors can also be used.



Generalization of Use Cases

- The specific use case represents a more specific behavior than the behavior of the general use case.
- Abstract use cases can also be used.



Process of Use Case Modeling

1. Capture a common vocabulary (glossary) / create domain model.
2. Name the system scope and boundaries.
3. Find and briefly describe actors; first primary then secondary.
4. For each actor determine a set of its use cases – “candidate use case list”.
5. Reconsider and revise use cases - add, subtract, merge use cases.
6. Briefly describe use cases.
7. Package use cases and actors.
8. Present the use case model in use case diagrams.
9. Describe use cases in details; brainstorm scenarios and analyze all attributes.
10. Structure the use case model.
11. Repeat the process until the model is complete and consistent.
12. Review the use case model.

Rules for Use Case Diagrams

- Differentiate primary and secondary actors.
- Use case must provide a real service to user.
- Keep drawings clear and neat.
  - Do not put too many use cases in one diagram.
  - Simpler diagrams are easier to understand.
- Split up the use case model into use case packages.
  - Package related use cases.
  - *Functional decomposition* of the system.