Analysis - dependencies

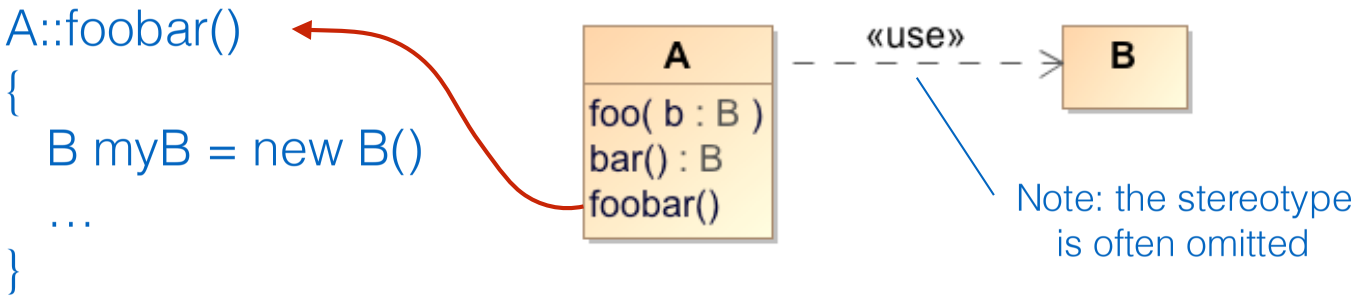
What is a dependency?

- "A dependency is a relationship between two elements where a change to one element (the supplier) may affect or supply information needed by the other element (the client)". In other words, the client *depends* in some way on the supplier
- It is a catch-all that is used to model several different types of relationship. We've already seen several types of dependency, «instantiate», «include» and «extend»
- There are three main types of dependency:
  - **Usage** - the client uses some of the services made available by the supplier to implement its own behavior – this is the most commonly used type of dependency
  - **Abstraction** - a shift in the level of abstraction. The supplier is more abstract than the client
  - **Permission** - the supplier grants some sort of permission for the client to access its contents – this is a way for the supplier to control and limit access to its contents

Usage dependencies

Usage dependency	Semantics
«use»	The client makes use of the supplier to implement its behavior
«call»	The client operation invokes the supplier operation
«parameter»	The supplier is a parameter of the client operation
«send»	The client operation sends the supplier Signal to some unspecified target
«instantiate»	The client is an instance of the supplier

«use» example



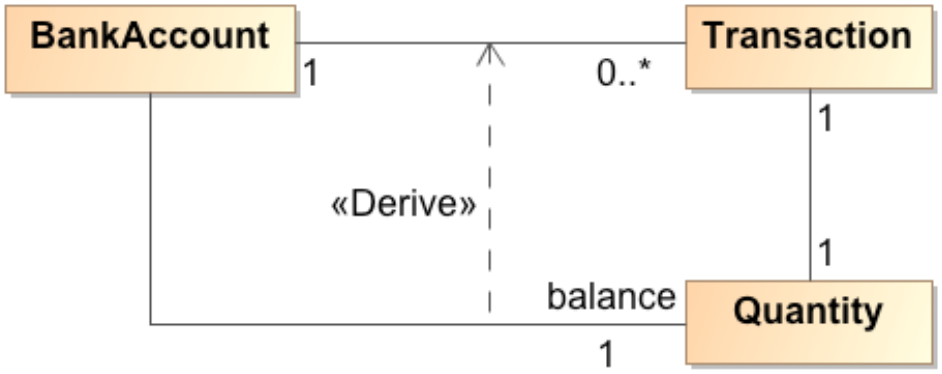
- A «use» dependency is generated between class A and B when:

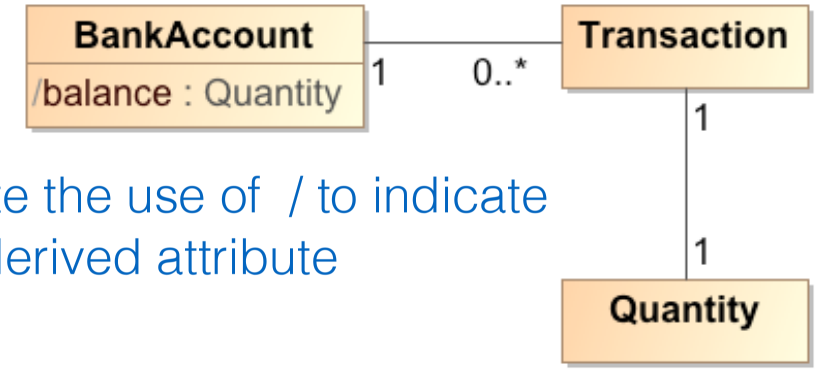
  1. An operation of class A needs a parameter of class B
  2. An operation of class A returns a value of class B
  3. An operation of class A uses an object of class B somewhere in its implementation

Abstraction dependencies

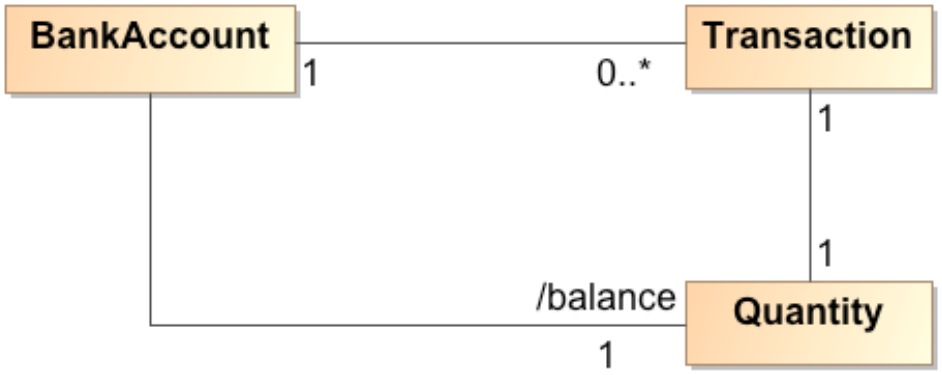
Abstraction dependency	Semantics
«Trace»	The client and the supplier represent the same concept but at different points in development
«Substitute»	The client may be substituted for the supplier at runtime. The client and supplier must realize a common contract. Use in environments that don't support specialization/ generalization
«Refine»	The client represents a fuller specification of the supplier
«Derive»	The client may be derived from the supplier. The client is logically redundant, but may appear for implementation reasons

«Derive» example





note the use of / to indicate a derived attribute



note the use of / to indicate a derived association end

- Three ways to express the same relationship using «Derive»

Permission dependencies

- More on this later when we look at packages…

Permission dependency	Semantics
«Access»	The public contents of the supplier package are added as private elements to the namespace of the client package
«Import»	The public contents of the supplier package are added as public elements to the namespace of the client package
«Permit»	The client element has access to the supplier element despite the declared visibility of the supplier

Summary

- Dependency is semantically the weakest type of relationship - it is used as a catch-all
- There are three types of dependency:
  - Usage
  - Abstraction
  - Permission