Object-Oriented Analysis and Design using JAVA

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

Lecture-16 Relationships --dependency

What is dependency relationship?

“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. We use dependencies to model relationships between classifiers where one classifier depends on the other in some way, but the relationship is not really an association.

Dependency relationship-For example, you may pass an object of one class as a parameter to a method of an object of a different class.

There is clearly some sort of relationship between the classes of those objects, but it is not really an association.

You can use the dependency relationship (specialized by certain predefined stereotypes) as a catch-all to model this kind of relationship.

UML 1.4 specifies four basic types of dependency

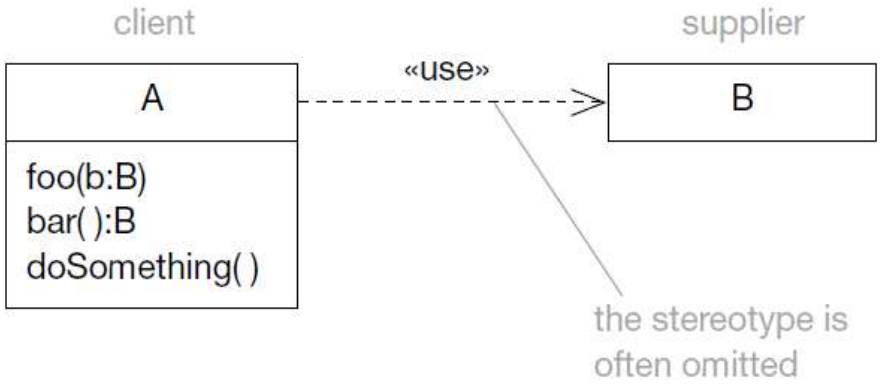
Type	Semantics
Usage	The client uses some of the services made available by the supplier to implement its own behavior – this is the most commonly used type of dependency
Abstraction	This indicates a relationship between client and supplier, where the supplier is more abstract than the client What do we mean by "more abstract"? This could mean that the supplier is at a different point in development than the client (e.g. in the analysis model rather than the design model)
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

Dependencies don’t just occur between classes. They can commonly occurbetween:

- packages and packages;
- objects and classes.

They can also occur between an operation and a class, although it is quite rare to show this explicitly on a diagram

Usage dependencies-The «use» dependency is the most common dependency between classes



Above figure shows two classes, A and B, that have a «use» dependency between them. This dependency is generated by any of the following cases.

- An operation of class A needs a parameter of class B.
- An operation of class A returns a value of class B.
- An operation of class A uses an object of class B somewhere in its implementation, but *not* as an attribute.

In previous PPT, Cases 1 and 2 are straightforward, but case 3 is more interesting. You would have this case if one of the methods of class A created a transient object of class B. A Java code fragment for this case is shown in following figure to illustrate the point.

```
Class A
{
   …
   void doSomething()
      {
         B myB = new B();
         // Use myB in some way
         …
      }
}
```

«call»
This dependency is between operations – the client operation invokes the supplier operation. This type of dependency tends not to be very widely used in UML modeling. It applies at a deeper level of detail than most modelers are prepared to go. Also, very few CASE tools currently support dependencies between operations.

«parameter»
This dependency is between an operation and a class. The supplier is a parameter or return value of an operation on the client. Again, this type of dependency tends not to be very widely used.

«send»
The client sends the supplier (which must be a signal) to some unspecified target.

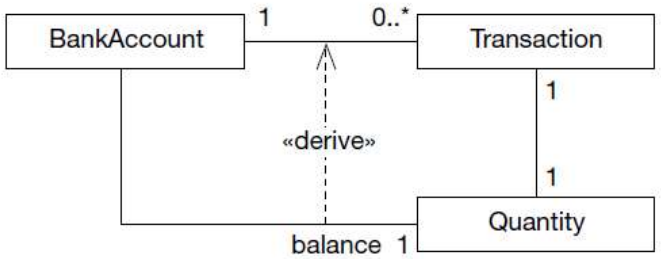
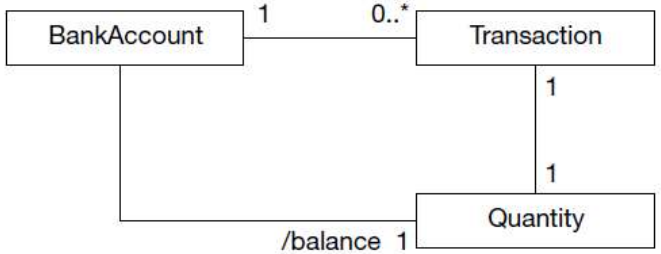
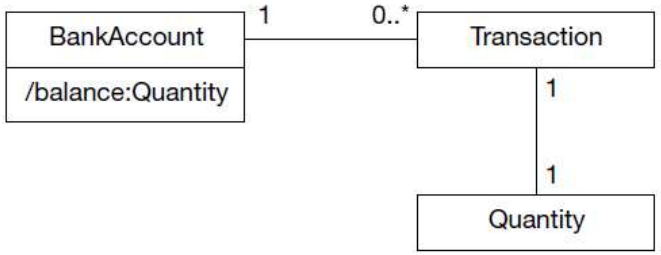
«instantiate»
The client is an instance of the supplier.

Abstraction dependencies- Abstraction dependencies model dependencies between things that are at different levels of abstraction. An example might be a class in an analysis model, and the same class in the design model.

«trace»
You often use a «trace» dependency to illustrate some general connection between elements where the supplier is at a different point in development to the client.

«refine»
Whereas the «trace» dependency is a purely historical dependency that is often between two elements in different models, «refine» may be between elements in the same model.

«derive»
You use this stereotype when you want to show explicitly that a thing can be derived in some way from some other thing.

Model	Description
	The BankAccount class has a derived association to Quantity where Quantity plays the role of the balance of the BankAccount This model emphasizes that the balance is derived from the BankAccount's collection of Transactions
	In this case a slash is used on the role name to indicate that the relationship between BankAccount and Quantity is derived This is less explicit as it does not show what the balance is derived from
	Here the balance is shown as a derived attribute – this is indicated by the slash that prefixes the attribute name This is the most concise expression of the dependency

Permission dependencies- Permission dependencies are about expressing the ability of one thing to access another thing.

«access»
This is a dependency between packages. Packages are used in UML to group things. The essential point here is that «access» allows one package to access all of the public contents of another package.

«import»
This is conceptually similar to «access» except that the namespace of the supplier is merged into the namespace of the client. This allows elements in the client to access elements in the supplier without having to qualify element names with the package name.

«friend»
This type of dependency allows a *controlled* violation of encapsulation, but on the whole it should be avoided. The client element has access to the supplier element, whatever the *declared* visibility of the supplier

Key references

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

Thank You