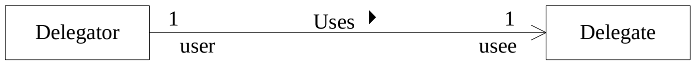
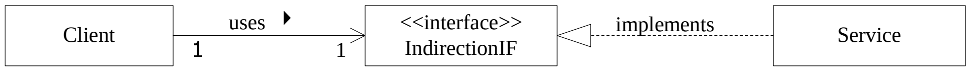
Delegation

Origin:	Grand 98
Reason:	An object needes to be different subclasses at different times.
Synopsis:	Extend a class *C* by writing an additional class *D* with added funtionality that uses instances of *C*.
Example:	An airline reservation system has roles such as *flight crew*, *passenger*, *ticket agent*,etc. A *person* can play different roles at different times, and more than one role at a time. Inheritance makes this impossible.
UML Class Diagram:	
Solution:	**Delegator** — Uses *Delegate*. — Flight crew, Passenger, etc. **Delegate** — Specializes *DelegateIF* to a specific role. — Person **DelegatorIF** — Base class or interface of all *Delegator*s. — Role
Note:	The example has two layers of delegation: *Role* to *Person* and *PersonWithRole* to *Role*.
Java API Usage:	It is the basis for Java's delegation event model.
See also:	Bridge, Decorator, Facade, Proxy

Interface

Origin:	Grand 98
Reason:	A class needs to be independent of services provided by another class.
Synopsis:	Abstract a class *C* by writing an interface *IF*. Clients access class *C* through interface *IF*.
Example:	A business application uses an *Address* class in a variety of objects, e.g., *Vendors*, *Clients*, *Freight companies*, etc. To make these objects less dependent of the details of the *Address* class, the objects should rather use an interface, *AddressIF*.
UML Class Diagram:	 Client uses 1 1 <<interface>> IndirectionIF implements Service
Solution:	**Service** — A class that provides data and/or methods to the *Client*. — Address **Client** — A class that uses the services of class *Service*. — Vendor, Client, etc. **ServiceIF** — Interface of class *Service*. — AddressIF
Java API Usage:	Ex: *Java.io.FilenameFilter* is an interface to tell if a named file is included a collection.