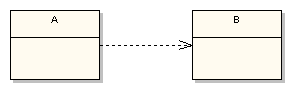
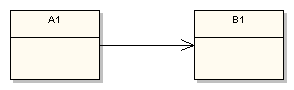
1. **Dependency:**

It means that the class at the source end of the relationship has some sort of dependency on the class at the target (arrowhead) end of the relationship. For example, the following simple states that class A depends on class B in some way:



1. **Association:**



Association defines dependency, but a much stronger dependency than that described above with the plain *dependency* relationship. The arrowhead means that there is a one-way relationship. In this example it means that class A1 is associated with class B1. In other words, class A1 uses and contains one instance of class B1, but B1 does not know about or contain any instances of class A1.

import B1;

public class A1 {

private B1 b1;

public B1 getB1() {

return b1;

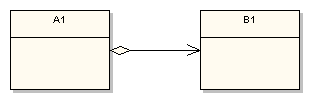
}

}

There are 2 types of associations, based on lifetimes of the instances that make up the dependent object’s state.

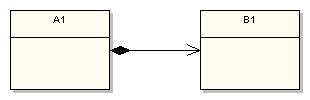
**Note: Multiplicity is defined only for aggregation and composition relationships.**

* 1. **Aggregation**



A clear diamond adornment has been added to the source side of the relationship. This means that A1 aggregates a B1. Aggregation describes an association where an instance of A1 contains a reference to an instance of B1 as part of the A1’s state, but the use of the specific instance of B1 is or may be shared among other aggregators. A shared association means that the lifetime of the aggregated object, the instance of B1 in this case, is outside the scope of the referencing object. Therefore, when a specific instance of A1 goes out of scope (e.g. garbage collected), the instance of B1 does not (of necessity) go out of scope.

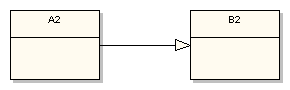
* 1. **Composition**



Composition on the other hand defines a relationship where the scope of the containing object (an A1) and the contained object (a B1) is related. When the containing object goes out of scope, then the contained object also goes out of scope.

1. **Generalization:**

UML generalization symbolizes what is known as inheritance in the world of object-oriented programming. It is sometimes also called specialization.



More specifically UML generalization corresponds to class extension in the Java language. The above diagram fragment would be implemented in Java as follows:

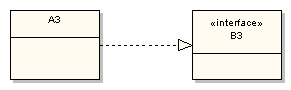
import B2;

public class A2 extends B2 { // . . . }

B2 is the super class and A2 is the subclass in the relationship. Just remember that the generalization symbol forms a line from the subclass to its super class with the clear triangular arrowhead pointing at the super class.

1. **Realization:**

This relationship is somewhat related to generalization, but a bit different. In object-oriented programming parlance realization represents the implementation of an interface by a class. So it represents how some characteristics of a class are defined, but says nothing about the implementation details:



This diagram fragment states that class A3 implements or realizes the interface defined by B3. In the Java language the above realization relationship would be programmed as follows:

import B3;

public class A3 implements B3 {

// . . .

}

Realization is very important when designing object-oriented subsystems and frameworks. The interface being realized in a class diagram represents a contract between the subsystem or framework and its consumer. The interface publisher guarantees that any consumer implementing one or more of its public interfaces properly will have some level of consistent integration with the interface-defining subsystem or framework.