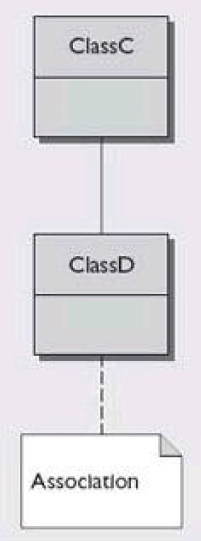
**Association:**

The association relationship is a way of describing that a class knows about and holds a reference to another class. This can be described as a "has-a" relationship because the typical implementation in Java is through the use of an instance field. The relationship can be bi-directional with each class holding a reference to the other. [Aggregation](http://java.about.com/od/a/g/aggregation.htm) and [composition](http://java.about.com/od/c/g/composition.htm) are types of association relationships.

**Bidirectional Association Example**

/\*\*

\* C contains reference of D. So, c is associated with D

\*/

class C {

private D myD;

public C(D d) {

this.myD = d;

}

}

/\*\*

\* D contains reference of C. So, d is associated with C

\*/

class D {

private C myC;

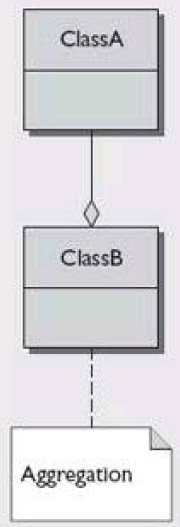
public D(C c) {

this.myC = c;

}

}

**Aggregation**

Aggregation is a relationship between two classes that is best described as a "has-a" and "whole/part" relationship. It is a more specialized version of the[association relationship](http://java.about.com/od/a/g/association.htm). The aggregate class contains a reference to another class and is said to have ownership of that class. Each class referenced is considered to be *part-of* the aggregate class

Ownership occurs because there can be no cyclic references in an aggregation relationship. If Class A contains a reference to Class B and Class B contains a reference to Class A then no clear ownership can be determined and the relationship is simply one of association.

For example, imagine a Student class that stores information about individual students at a school. Now let's say there is a Subject class that holds the details about a particular subject (e.g., history, geography). If the Student class is defined to contain a Subject object then it can be said that the Student object*has-a* Subject object. The Subject object also makes up *part-of* the Student object, after all there is no student without a subject to study. The Student object is therefore the owner of the Subject object.

**Examples:**

There is an aggregation relationship between Student class and the Subject class:

**class** Subject {

**private** String name;

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getName() {

**return** name;

}

}

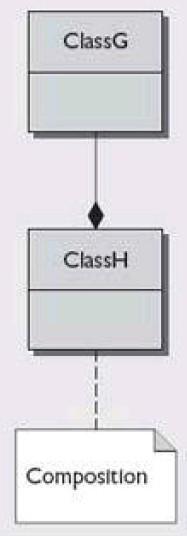
**class** Student {

**private** Subject[] studyAreas = **new** Subject[10];

// the rest of the Student class }

}

**Composition**

Composition is a relationship between two classes that is based on the [aggregation relationship](http://java.about.com/od/a/g/aggregation.htm). Composition takes the relationship one step further by ensuring that the containing object is responsible for the lifetime of the object it holds. If Object B is contained within Object A, then Object A is responsible for the creation and destruction of Object B. Unlike aggregation, Object B cannot exist without Object A.

**Examples:**

Imagine you create a Student class that holds information about individual students at a school. One piece of information stored is the student's date of birth. It's held in a GregorianCalendar object:

import java.util.GregorianCalendar;

public class Student {

  private String name;

  private GregorianCalendar dateOfBirth;

  public Student(String name, int day, int month, int year)

  {

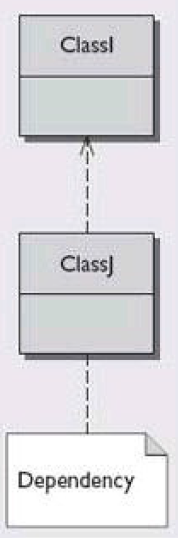
    this.name = name;

    this.dateOfBirth = new GregorianCalendar(year, month, day);

} //rest of Student class..

}

As the student class is responsible for the creation of the GregorianCalendar object it will also be responsible for its destruction (i.e., once the Student object no longer exists neither will the GregorianCalendar object). Therefore the relationship between the two classes is composition because Student has-a GregorianCalendar and it also controls its lifetime. The GreogrianCalender object cannot exist without the Student object.

**Dependency**

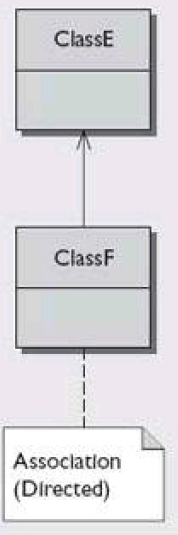
Dependency association depicts one class having a temporary association with another class.

Dependency associations occur when a class needs another class to exist or when an object is used as

a return value, local variable, or method argument. Dependency is depicted with a dotted line and a

stick arrow.

**Directed Association**

An association has (directed) navigation when it is marked with a navigability arrow, also described

as a stick arrow. This directed association’s arrow denotes navigation in the direction of that end, the

classifier has ownership of the marked association end, and the unmarked association’s end is owned by the association. In addition to the navigability arrow, directed association is depicted with a solid line.