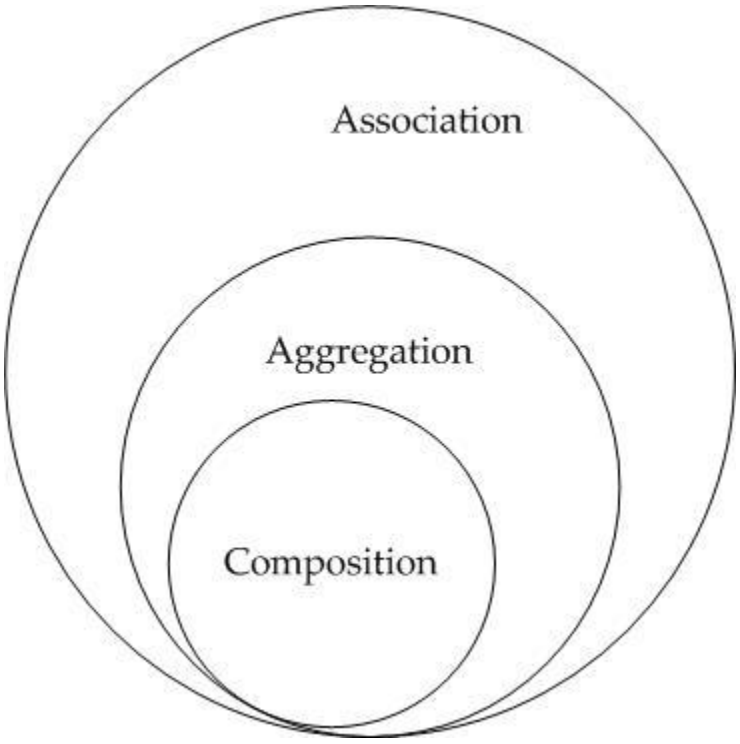
When we think of Object-oriented nature we always think of Objects, class (objects blueprints) and the relationship between them. Objects are related and interact with each other via methods. In other words the object of one class may use services/methods provided by the object of another class. This kind of relationship is termed as association..

Aggregation and Composition are subsets of association meaning they are specific cases of association.



- **I**n both aggregation and composition object of one class "owns" object of another class.
- But there is a subtle difference. In Composition the object of class that is owned by the object of it's owning class cannot live on it's own(Also called "death relationship"). It will always live as a part of it's owning object where as in Aggregation the dependent object is standalone and can exist even if the object of owning class is dead.
- So in composition if owning object is garbage collected the owned object will also be which is not the case in aggregation.

Confused?

Composition Example : Consider the example of a Car and an engine that is very specific to that car (meaning it cannot be used in any other car). This type of relationship between Car and SpecificEngine class is called Composition. An object of the Car class cannot exist

without an object of SpecificEngine class and object of **SpecificEngine has no** significance without Car class. To put in simple words Car class solely "owns" the SpecificEngine class.

Aggregation Example : Now consider class Car and class Wheel. Car needs a Wheel object to function. Meaning the Car object owns the Wheel object but we cannot say the Wheel object has no significance without the Car Object. It can very well be used in a Bike, Truck or different Cars Object.

Summing it up -

To sum it up association is a very generic term used to represent when a class uses the functionalities provided by another class. We say it's composition if one parent class object owns another child class object and that child class object cannot meaningfully exist without the parent class object. If it can then it is called Aggregation.

Association is generalized concept of relations between objects.

Composition(*mixture*) is a way to wrap simple objects or data types into a single unit. Compositions are a critical building block of many basic data structures

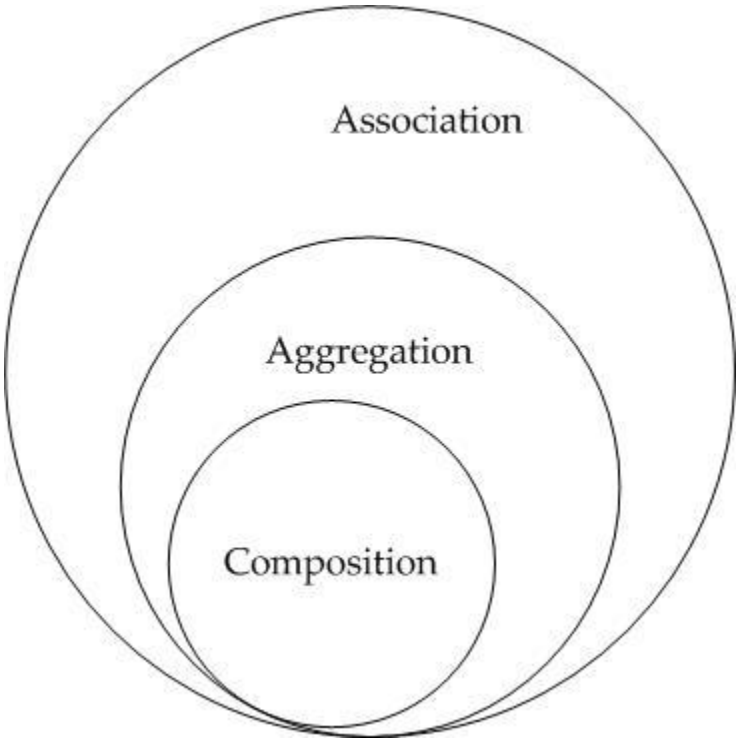
Aggregation(*The formation of a number of things into a cluster*) differs from ordinary composition in that it does not imply ownership. In composition, when the owning object is destroyed, so are the contained objects. In aggregation, this is not necessarily true.

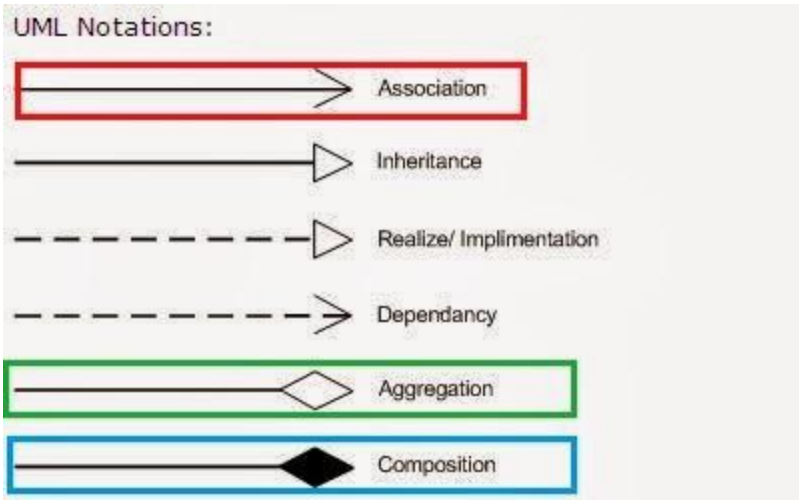
Trick to remember the difference :

- **"can-call": Association**
- **"Has-A": Aggregation**
- **"Part-Of": comPOsition**

context	Aggregation	Composition
Lifetime	objects have their own lifetime and there is no owner	controlled by whole or parent that owns it
Scope	parent objects and child objects are independent	parent object also means the death of its children.
Relationship	Has-a	Part-of
Strength	weak relationship	strong relationship.
Real-life example	Car and Driver	Car and wheels

- **Now let observe the following image**





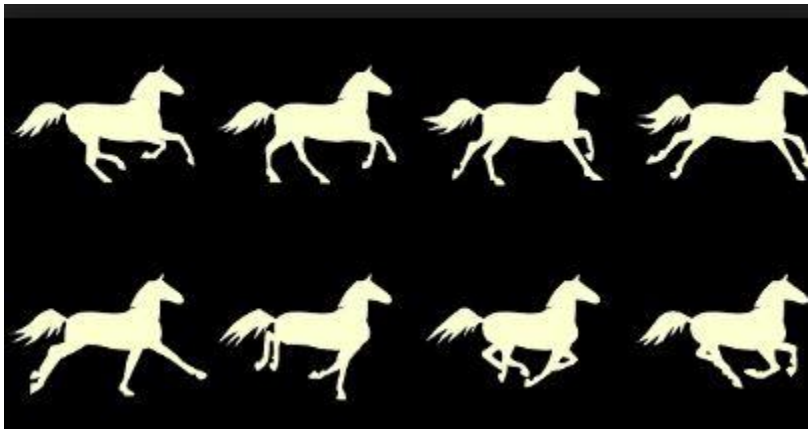
Association / Aggregation / Composition (Tight Coupling): Use a solid line with an association arrow between the classes. This represents a general relationship between classes, and changes in one class can directly impact the other. You can also use association end multiplicity to indicate the nature and cardinality of the relationship.

Dependency / Interface or Abstract Class (Loose Coupling): You can use a dashed line with an arrow pointing from one class to another. This indicates that one class depends on the other in some way. Dependencies can be tight if changes in the supplier class directly affect the client class

Analogy:

***Composition*:** The following picture is image composition i.e. using individual images making one image.

Aggregation **: collection of image in single location**



For example, A university owns various departments, and each department has a number of professors. If the university closes, the departments will no longer exist, but the professors in those departments will continue to exist. Therefore, a University can be seen as a composition of departments, whereas departments have an aggregation of professors. In addition, a Professor could work in more than one department, but a department could not be part of more than one university.

Dependency (references)
It means there is no conceptual link between two objects. e.g. EnrollmentService object references Student & Course objects (as method parameters or return types)

```
public class EnrollmentService { public void enroll(Student s, Course c){} }
```

Association (has-a)
It means there is almost always a link between objects (they are associated). Order object ***has a*** Customer object

```
public class Order { private Customer customer }
```

Aggregation (has-a + whole-part)
A special kind of association where there is whole-part relation between two objects. they might live without each other though.

```
public class PlayList { private List<Song> songs; }
```

OR

```
public class Computer { private Monitor monitor; }
```

Note: the trickiest part is to distinguish aggregation from normal association. Honestly, I think this is open to different interpretations.

Composition (has-a + whole-part + ownership)
A special kind of aggregation. An `Apartment` is composed of some `Rooms`. A `Room` cannot exist without an `Apartment`. when an apartment is deleted, all associated rooms are deleted as well.

public class Apartment{ private Room bedroom; public Apartment() { bedroom = new Room(); } }

or two objects, `Foo` and `Bar` the relationships can be defined

Association - I have a relationship with an object. `Foo` uses `Bar`

public class Foo { private Bar bar; };
NB: See Fowler's definition - the key is that `Bar` is semantically related to `Foo` rather than just a dependency (like an `int` or `string`).

Composition - I own an object and I am responsible for its lifetime. When `Foo` dies, so does `Bar`

public class Foo { private Bar bar = new Bar(); }
Aggregation - I have an object which I've borrowed from someone else. When `Foo` dies, `Bar` may live on.

```
public class Foo {
        private Bar bar;
        Foo(Bar bar) {
        this.bar = bar;
        }
}
```

Must read: Object Composition-Delegation in C++ with Examples - GeeksforGeeks