# Association, Aggregation, and Composition

These terms signify the relationships between classes.

Aggregation is just a Collection or an array of elements of a single type. Association refers to either is-a/inheritance or composition/has-a relationships.

**Association** is a relationship where all object have their own lifecycle and there is no owner. Let's take an example of Teacher and Student. Multiple students can associate with single teacher and single student can associate with multiple teachers but there is no ownership between the objects and both have their own lifecycle. Both can create and delete independently.

**Aggregation** is a specialize form of Association where all object have their own lifecycle but there is ownership and child object can not belongs to another parent object. Let's take an example of Department and teacher. A single teacher can not belongs to multiple departments, but if we delete the department teacher object will not destroy. We can think about "has-a" relationship.

We can express aggregation in java as:

```
public class Address {

              ......

                 }



public class Person {

    private Address address;

    public Person(Address  address){

          this.address = address;

   }

}
```

here address can exist without a person.

**Composition** is again specialize form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object doesn't have their lifecycle and if parent object deletes all child object will also be deleted. Let's take again an example of relationship between House and rooms. House can contain multiple rooms there is no independent life of room and any room can't belongs to two different house if we delete the house room will automatically delete. Let's take another example relationship between Questions and options. Single questions can have multiple options and option can not belong to multiple questions. If we delete questions options will automatically delete. Composition can be representing with "part-of" relationship.

so how we can express composition relationship in Java:-

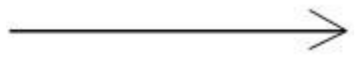
let see this example

```
public class Car {
                                final Engine engine;
                                public Car (){
                                              engine = new Engine (); // always having engine
                                             }

                                  }
```

## Association

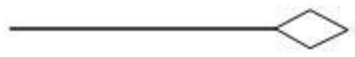
Association is a relationship between two objects. In other words, association defines the multiplicity between objects. You may be aware of one-to-one, one-to-many, many-to-one, many-to-many all these words define an association between objects. Aggregation is a special form of association. Composition is a special form of aggregation.



***Example:*** A Student and a Faculty are having an association.

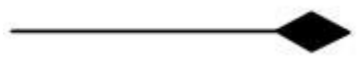
## Aggregation

Aggregation is a special case of association. A directional association between objects. When an object 'has-a' another object, then you have got an aggregation between them. Direction between them specified which object contains the other object. Aggregation is also called a "Has-a" relationship.



## Composition

Composition is a special case of aggregation. In a more specific manner, a restricted aggregation is called composition. When an object contains the other object, if the contained object cannot exist without the existence of container object, then it is called composition.



***Example:*** A class contains students. A student cannot exist without a class. There exists composition between class and students.

## Difference between aggregation and composition

Composition is more restrictive. When there is a composition between two objects, the composed object cannot exist without the other object. This restriction is not there in aggregation. Though one object can contain the other object, there is no condition that the composed object must exist. The existence of the composed object is entirely optional. In both aggregation and composition, direction is must. The direction specifies, which object contains the other object.

***Example:*** A Library contains students and books. Relationship between library and student is aggregation. Relationship between library and book is composition. A student can exist without a library and therefore it is aggregation. A book cannot exist without a library and therefore its a composition. For easy understanding I am picking this example. Don't go deeper into example and justify relationships!

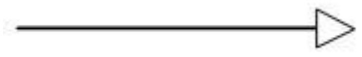
## Abstraction

Abstraction is specifying the framework and hiding the implementation level information. Concreteness will be built on top of the abstraction. It gives you a blueprint to follow to while implementing the details. Abstraction reduces the complexity by hiding low level details.

***Example:*** A wire frame model of a car.

## Generalization

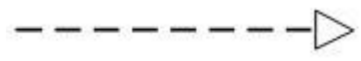
Generalization uses a "is-a" relationship from a specialization to the generalization class. Common structure and behaviour are used from the specializtion to the generalized class. At a very broader level you can understand this as inheritance. Why I take the term inheritance is, you can relate this term very well. Generalization is also called a "Is-a" relationship.



***Example:*** Consider there exists a class named Person. A student is a person. A faculty is a person. Therefore here the relationship between student and person, similarly faculty and person is generalization.

### Realization

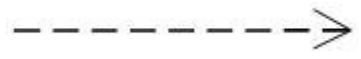
Realization is a relationship between the blueprint class and the object containing its respective implementation level details. This object is said to realize the blueprint class. In other words, you can understand this as the relationship between the interface and the implementing class.



***Example:*** A particular model of a car 'GTB Fiorano' that implements the blueprint of a car realizes the abstraction.

### Dependency

Change in structure or behaviour of a class affects the other related class, then there is a dependency between those two classes. It need not be the same vice-versa. When one class contains the other class it this happens.



***Example:*** Relationship between shape and circle is dependency

---

### Extracting real world relationships from a requirement

The whole point of OOP is that your code replicates real world objects, thus making your code readable and maintainable. When we say real world, the real world has relationships. Let's consider the simple requirement listed below:

1. Manager is an employee of XYZ limited corporation.
2. Manager uses a swipe card to enter XYZ premises.
3. Manager has workers who work under him.
4. Manager has the responsibility of ensuring that the project is successful.
5. Manager's salary will be judged based on project success.

If you flesh out the above five point requirement, we can easily visualize four relationships:-

- Inheritance
- Aggregation
- Association

- Composition

Let’s understand them one by one.

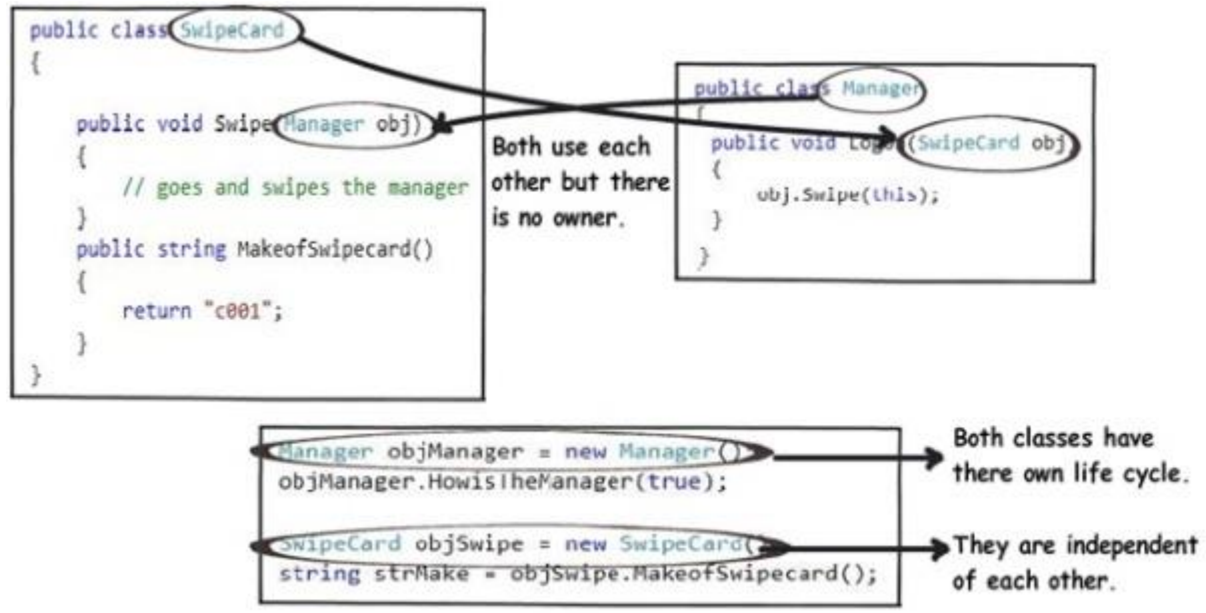
### Requirement 1: The IS A relationship

If you look at the first requirement (Manager is an employee of XYZ limited corporation), it’s a parent child relationship or inheritance relationship. The sentence above specifies that Manager is a type of employee, in other words we will have two classes: parent class Employee, and a child class Manager which will inherit from the Employee class.

**Note**: The scope of this article is only limited to aggregation, association, and composition. We will not discuss inheritance in this article as it is pretty straightforward and I am sure you can get 1000s of articles on the net which will help you in understanding it.

### Requirement 2: The Using relationship: Association

Requirement 2 is an interesting requirement (Manager uses a swipe card to enter XYZ premises). In this requirement, the manager object and the swipe card object use each other but they have their own object life time. In other words, they can exist without each other. The most important point in this relationship is that there is no single owner.



```
public class SwipeCard
{
    public void Swipe(Manager obj)
    {
        // goes and swipes the manager
    }
    public string MakeofSwipecard()
    {
        return "c001";
    }
}
```

```
public class Manager
{
    public void Login(SwipeCard obj)
    {
        obj.Swipe(this);
    }
}
```

```
Manager objManager = new Manager();
objManager.HowIsTheManager(true);

SwipeCard objSwipe = new SwipeCard();
string strMake = objSwipe.MakeofSwipecard();
```

Both use each other but there is no owner.
Both classes have there own life cycle.
They are independent of each other.

**Association**

The above diagram shows how the SwipeCard class uses the Manager class and the Manager class uses the SwipeCard class. You can also see how we can create objects of the Manager class and SwipeCard class independently and they can have their own object life time.

This relationship is called the “Association” relationship.

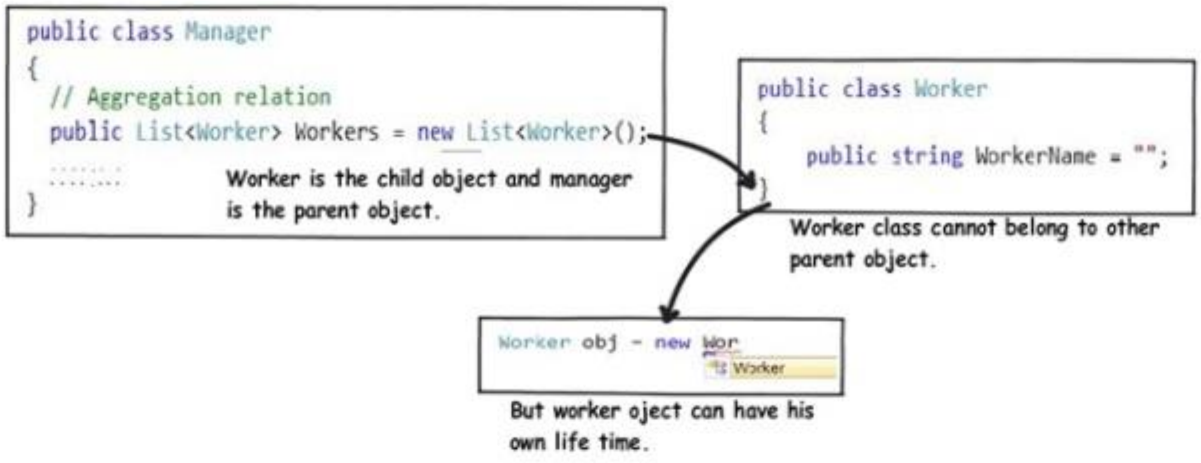
## Requirement 3: The Using relationship with Parent: Aggregation

The third requirement from our list (Manager has workers who work under him) denotes the same type of relationship like association but with a difference that one of them is an owner. So as per the requirement, the Manager object will own Worker objects.

The child Worker objects can not belong to any other object. For instance, a Worker object cannot belong to a SwipeCard object.

But… the Worker object can have its own life time which is completely disconnected from the Manager object. Looking from a different perspective, it means that if the Manager object is deleted, the Worker object does not die.

This relationship is termed as an "Aggregation" relationship.



```
public class Manager
{
  // Aggregation relation
  public List<Worker> Workers = new List<Worker>();
  ........
}
```

Worker is the child object and manager is the parent object.

```
public class Worker
{
    public string WorkerName = "";
}
```

Worker class cannot belong to other parent object.

```
Worker obj = new Wor
```

But worker oject can have his own life time.

Aggregation

## Requirements 4 and 5: The Death relationship: Composition

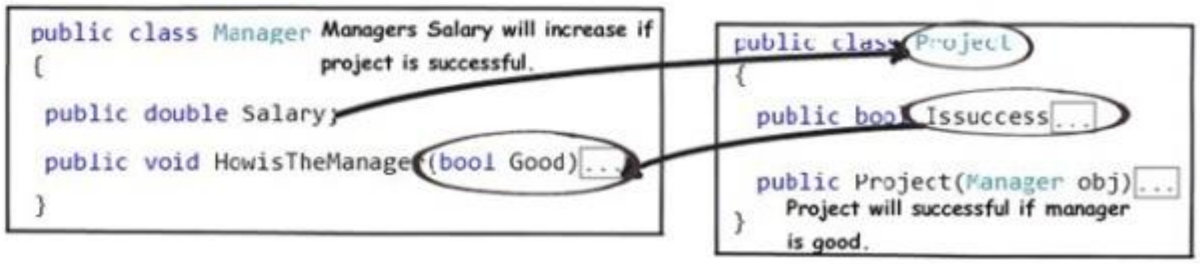
The last two requirements are actually logically one. If you read closely, the requirements are as follows:

1. Manager has the responsibility of ensuring that the project is successful.
2. Manager's salary will be judged based on project success.

Below is the conclusion from analyzing the above requirements:

1. Manager and the project objects are dependent on each other.
2. The lifetimes of both the objects are the same. In other words, the project will not be successful if the manager is not good, and the manager will not get good increments if the project has issues.

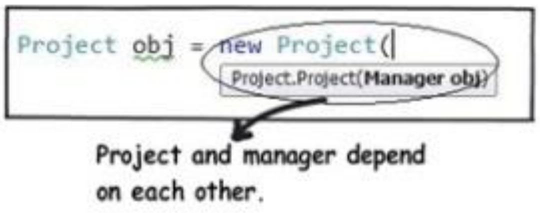
Below is how the class formation will look like. You can also see that when I go to create the project object, it needs the manager object.



```
public class Manager
{
 public double Salary;
 public void HowisTheManager(bool Good)...
}
```

```
public class Project
{
 public bool Issuccess...
 public Project(Manager obj)...
}
```

Managers Salary will increase if project is successful.
Project will successful if manager is good.

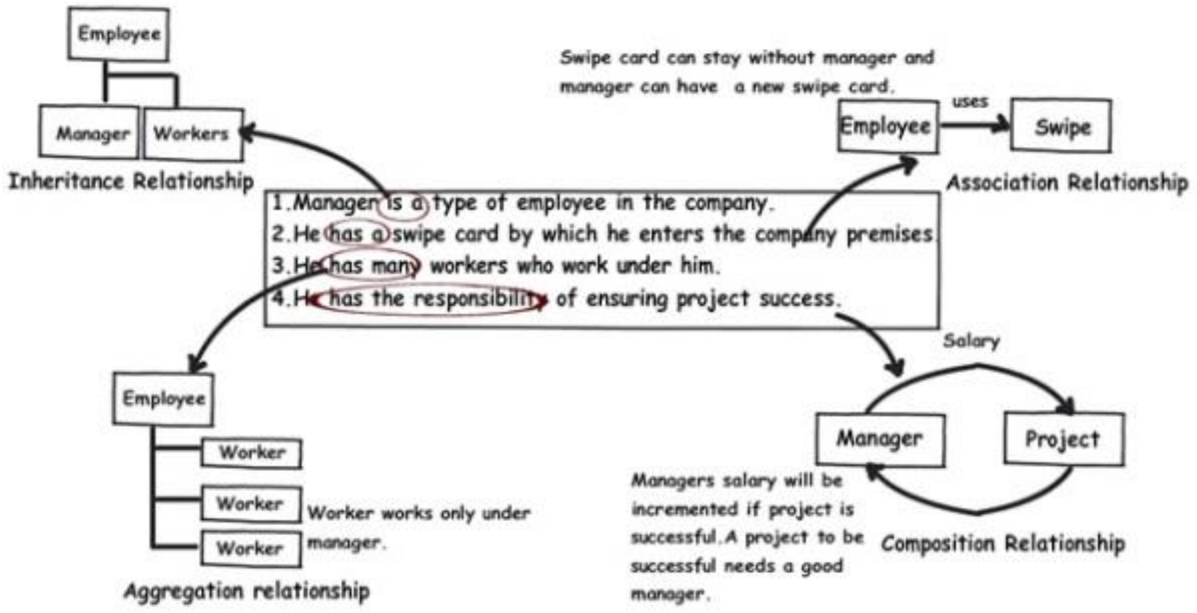


```
Project obj = new Project(
Project.Project(Manager obj)
```

This relationship is termed as the composition relationship. In this relationship, both objects are heavily dependent on each other. In other words, if one goes for garbage collection the other also has to be garbage collected, or putting from a different perspective, the lifetime of the objects are the same. That's why I have put in the heading "Death" relationship.

## Putting things together

Below is a visual representation of how the relationships have emerged from the requirements.



1. Manager is a type of employee in the company.
2. He has a swipe card by which he enters the company premises.
3. He has many workers who work under him.
4. He has the responsibility of ensuring project success.

Employee
Manager
Workers
Inheritance Relationship
Swipe card can stay without manager and manager can have a new swipe card.
Employee
uses
Swipe
Association Relationship
Employee
Worker
Worker
Worker
Worker works only under manager.
Aggregation relationship
Salary
Manager
Project
Managers salary will be incremented if project is successful. A project to be successful needs a good manager.
Composition Relationship

## The source code

You can download the sample source code for this article.

## Summarizing

To avoid confusion henceforth for these three terms, I have put forward a table below which will help us compare them from three angles: owner, lifetime, and child object.

| | **Association** | **Aggregation** | **Composition** |
|---|---|---|---|
| **Owner** | No owner | Single owner | Single owner |
| **Life time** | Have their own lifetime | Have their own lifetime | Owner's life time |
| **Child object** | Child objects all are independent | Child objects belong to a single parent | Child objects belong to a single parent |

## Video