# OOPs concepts – What is Aggregation in java?

BY CHAITANYA SINGH | FILED UNDER: [OOPS CONCEPT](https://beginnersbook.com/category/oops-concept/)

Aggregation is a special form of association. It is a relationship between two classes like [association](https://beginnersbook.com/2013/05/association/), however its a **directional** association, which means it is strictly a **one way association.** It represents a **HAS-A** relationship.

## Aggregation Example in Java

For example consider two classes Student class and Address class. Every student has an address so the relationship between student and address is a Has-A relationship. But if you consider its vice versa then it would not make any sense as an Address doesn’t need to have a Student necessarily. Lets write this example in a java program.  
Student Has-A Address

class Address

{

int streetNum;

String city;

String state;

String country;

Address(int street, String c, String st, String coun)

{

this.streetNum=street;

this.city =c;

this.state = st;

this.country = coun;

}

}

class StudentClass

{

int rollNum;

String studentName;

//Creating HAS-A relationship with Address class

Address studentAddr;

StudentClass(int roll, String name, Address addr){

this.rollNum=roll;

this.studentName=name;

this.studentAddr = addr;

}

public static void main(String args[]){

Address ad = new Address(55, "Agra", "UP", "India");

StudentClass obj = new StudentClass(123, "Chaitanya", ad);

System.out.println(obj.rollNum);

System.out.println(obj.studentName);

System.out.println(obj.studentAddr.streetNum);

System.out.println(obj.studentAddr.city);

System.out.println(obj.studentAddr.state);

System.out.println(obj.studentAddr.country);

}

}

Output:

123

Chaitanya

55

Agra

UP

India

The above example shows the **Aggregation** between Student and Address classes. You can see that in Student class I have declared a property of type Address to obtain student address. Its a typical example of Aggregation in Java.

## Why we need Aggregation?

**To maintain code re-usability**. To understand this lets take the same example again. Suppose there are two other classes College and Staff along with above two classes Student and Address. In order to maintain Student’s address, College Address and Staff’s address we don’t need to use the same code again and again. We just have to use the reference of Address class while defining each of these classes like:

Student Has-A Address (Has-a relationship between student and address)

College Has-A Address (Has-a relationship between college and address)

Staff Has-A Address (Has-a relationship between staff and address)

Hence we can improve code re-usability by using Aggregation relationship.

So if I have to write this in a program, I would do it like this:

class Address

{

int streetNum;

String city;

String state;

String country;

Address(int street, String c, String st, String coun)

{

this.streetNum=street;

this.city =c;

this.state = st;

this.country = coun;

}

}

class StudentClass

{

int rollNum;

String studentName;

//Creating HAS-A relationship with Address class

Address studentAddr;

StudentClass(int roll, String name, Address addr){

this.rollNum=roll;

this.studentName=name;

this.studentAddr = addr;

}

...

}

class College

{

String collegeName;

//Creating HAS-A relationship with Address class

Address collegeAddr;

College(String name, Address addr){

this.collegeName = name;

this.collegeAddr = addr;

}

...

}

class Staff

{

String employeeName;

//Creating HAS-A relationship with Address class

Address employeeAddr;

Staff(String name, Address addr){

this.employeeName = name;

this.employeeAddr = addr;

}

...

}

As you can see that we didn’t write the Address code in any of the three classes, we simply created the HAS-A relationship with the Address class to use the Address code. The dot dot(…) part in the above code can be replaced with the public static void main method, the code in it would be similar to what we have seen in the first example.

**OOPs concepts – What is Association in java?**

BY CHAITANYA SINGH | FILED UNDER: [OOPS CONCEPT](https://beginnersbook.com/category/oops-concept/)

In this article we will discuss **Association in Java**. Association establishes relationship between two separate **classes** through their **objects**. The relationship can be one to one, One to many, many to one and many to many.

**Association Example**

class CarClass{

String carName;

int carId;

CarClass(String name, int id)

{

this.carName = name;

this.carId = id;

}

}

class Driver extends CarClass{

String driverName;

Driver(String name, String cname, int cid){

super(cname, cid);

this.driverName=name;

}

}

class TransportCompany{

public static void main(String args[])

{

Driver obj = new Driver("Andy", "Ford", 9988);

System.out.println(obj.driverName+" is a driver of car Id: "+obj.carId);

}

}

Output:

Andy is a driver of car Id: 9988

In the above example, there is a one to one relationship(**Association**) between two classes: CarClass and Driver. Both the classes represent two separate entities.

**Association vs Aggregation vs Composition**

Lets discuss **difference between Association, Aggregation and Composition**:

Although all three are related terms, there are some major differences in the way they relate two classes. **Association** is a relationship between two separate classes and the association can be of any type say one to one, one to may etc. It joins two entirely separate entities.

[Aggregation](https://beginnersbook.com/2013/05/aggregation/) is a special form of association which is a unidirectional one way relationship between classes (or entities), for e.g. Wallet and Money classes. Wallet has Money but money doesn’t need to have Wallet necessarily so its a one directional relationship. In this relationship both the entries can survive if other one ends. In our example if Wallet class is not present, it does not mean that the Money class cannot exist.

**Composition** is a restricted form of Aggregation in which two entities (or you can say classes) are highly dependent on each other. For e.g. Human and Heart. A human needs heart to live and a heart needs a Human body to survive. In other words when the classes (entities) are dependent on each other and their life span are same (if one dies then another one too) then its a composition. Heart class has no sense if Human class is not present.