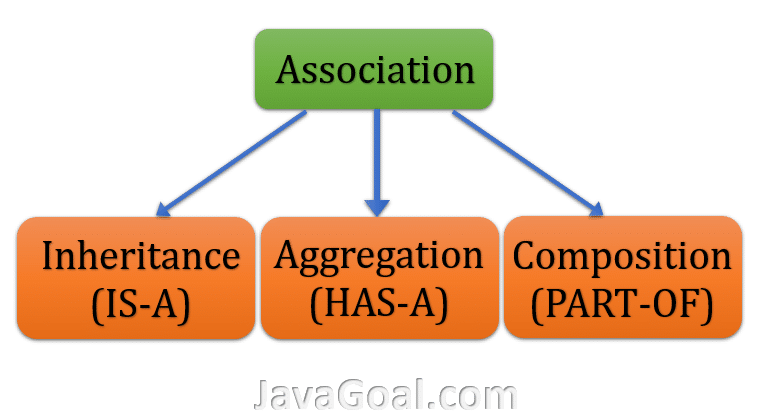
**Types of Association in Java**

1. ***Inheritance (Is – A relationship)***
2. ***Aggregation (has-A relationship)***
3. ***Composition (Part of relationship)***



[**1. Inheritance in Java(Is – A)**](https://javagoal.com/java-inheritance/)

Inheritance is an important concept/feature of Object-Oriented Programming. Inheritance is a  mechanism by which one class can acquire/inherit the features(fields and methods) of another class. In this article, we will read the concept of Inheritance in java**.**It means in java one class can inherit the variables and methods of another class. It is also known as the **Is-A** **relationship**. You can read [**inheritance in Java**](https://javagoal.com/java-inheritance/) in detail.

**2.** **Aggregation in java(Has-A)**

* **Aggregation** represents the Has-A relationship.
* It is a one-way relationship and called unidirectional association. For example, Bank can have employees but vice versa is not possible and that’s why it unidirectional in nature.
* In JAVA Aggregation,  both the class’s object will not affect each other.

**Example of aggregation( Has-A):** Let’s take the example of a **Bank**and a customer. A **customer**belongs to**Bank.**If a **customer**is not coming to the **bank**, the **bank**will not be deleted because it may still be functional. We are taking an example of a **Bank**and a **customer**. It will show the **HAS-A relationship** between **Bank**and **customer**. A bank **HAS-A** customer.

**class** Bank

{

**String** nameOfBank;;

**int** IFSC;

Bank(**String** nameOfBank, **int** IFSC)

{

**this**.nameOfBank = nameOfBank;

**this**.IFSC = IFSC;

}

**public** **void** displayAllDetails(Customer customer)

{

System.out.println("Name of Bank = "+ nameOfBank);

System.out.println("IFSC of Bank = "+ IFSC);

System.out.println("Name of Customer = "+ customer.nameOfCustomer);

System.out.println("Account number of Customer = "+ customer.accountNumber);

System.out.println("Amount of Customer = "+ customer.amount);

}

}

**class** Customer

{

**String** nameOfCustomer;

**int** accountNumber;

**int** amount;

Customer(**String** nameOfCustomer, **int** accountNumber, **int** amount)

{

**this**.nameOfCustomer = nameOfCustomer;

**this**.accountNumber = accountNumber;

**this**.amount = amount;

}

}

**class** Branch

{

**public** **static** **void** main(**String** arg[])

{

Bank bank = new Bank("AXIS", 12345);

Customer customer = new Customer("Ram", 123456789, 100000);

bank.displayAllDetails(customer);

}

}

***Output:*** *Name of Bank = AXIS  
IFSC of Bank = 12345  
Name of Customer = Ram  
Account number of Customer = 123456789  
Amount of Customer = 100000*

In this example, we have two classes **Bank** and **Customer**. We need only some information from the **Customer**. So, we have created an object of Customer and send it as an argument. After that, we used the object of the Customer.

**3. Composition in java(Part of)**

It is a restricted form of Aggregation. In composition two entities are highly dependent on each other. One entity cannot exist without the other.

* It represents a **part-of relationship**.
* In composition, both the entities are dependent on each other.
* It is a highly restricted form.

**Example of Composition( Part – Of):** For example, a **car** has an **engine**. Composition makes strong relationship between the objects. It means that if we destroy the owner object, its members also will be destroyed with it. For example, if the Car is destroyed the engine is destroyed as well.

**class** Car

{

//final will make sure engine is initialized

**private** **final** Engine engine;

**String** nameOfCar;

**String** modell;

**public** Car(**String** nameOfCar, **String** model)

{

engine = new Engine("POWERHIGH", "ABC");

**this**.nameOfCar = nameOfCar;

**this**.model = model;

}

**public** **void** showAlldetails()

{

System.out.println("Name of Car ="+nameOfCar);

System.out.println("Name of Model ="+model);

System.out.println("Engine used in Car ="+engine.typeOfEngine);

System.out.println("Engine name of Car ="+engine.nameOfEngine);

}

}

**class** Engine

{

**String** typeOfEngine;

**String** nameOfEngine;

Engine(**String** typeOfEngine, **String** nameOfEngine)

{

**this**.typeOfEngine = typeOfEngine;

**this**.nameOfEngine = nameOfEngine;

}

}

**class** ShowRoom

{

**public** **static** **void** main(**String** arg[])

{

Car car = new Car("BMW", "5X");

car.showAlldetails();

}

}

***Output:*** *Name of Car =BMW  
Name of Model =5X  
Engine used in Car =POWERHIGH  
Engine name of Car =ABC*

In this example, we have two classes **Car** and **Engine**. The Engine class is totally depending on Car. Because Engine is a **Part-Of** car. Without an engine, the car is nothing. So, in JAVA Car class we created an object of Engine class which initialized in the Car class. It is a highly dependent class.

**Delegation is simply passing a duty off to someone/something else.**

* **Delegation can be an alternative to inheritance.**
* **Delegation means that you use an object of another class as an instance variable, and forward messages to the instance.**

**/ Java program to illustrate**

**// delegation**

**class RealPrinter {**

**// the "delegate"**

**void print()**

**{**

**System.out.println("The Delegate");**

**}**

**}**

**class Printer {**

**// the "delegator"**

**RealPrinter p = new RealPrinter();**

**// create the delegate**

**void print()**

**{**

**p.print(); // delegation**

**}**

**}**

**public class Tester {**

**// To the outside world it looks like Printer actually prints.**

**public static void main(String[] args)**

**{**

**Printer printer = new Printer();**

**printer.print();**

**}**

**}**