| Quick Links for Java |
| --- |
| Recent Articles |
| MCQ / Quizzes |
| Java Collections |
| Practice Problems |
| Commonly Asked Questions Set 1 & Set 2 |

| Basics |
| --- |
| Identifiers, Data types & Variables |
| Scope of Variables |
| Operators |
| Loops and Decision Making |
| Explore More... |
| **Input / Output** |
| Ways to read Input from Console |
| Scanner VS BufferReader Class |
| Formatted output |

- Fast I/O in Java in Competitive Programming
- Command Line arguments
- Explore More...

**Arrays**

- Arrays in Java
- Default array values in Java
- Compare two arrays
- Final Arrays & Jagged Arrays
- Array IndexOutofbounds Exception
- Explore More...

**Strings**

- String Class in Java
- StringBuffer , StringTokenizer & StringJoiner
- Initialize and Compare Strings
- String vs StringBuilder vs StringBuffer
- Integer to String & String to Integer
- Search, Reverse and Split()
- Explore More...

**OOP in Java**

- Classes and Objects in Java
- Different ways to create objects

- Access Modifiers in Java
- Object class in Java
- Encapsulation & Inheritance
- Method Overloading & Overriding
- Explore More...

**Constructors**

- Constructors & Constructor Chaining
- Constructor Overloading
- Private Constructors and Singleton Classes
- Explore More...

**Methods**

- Parameter Passing
- Returning Multiple Values
- Private and Final Methods
- Default Methods
- Explore More...

**Exception Handling**

- Exceptions & Types of Exceptions
- Flow control in try-catch & Multicatch
- throw and throws

Explore More...

**Multithreading**

Multithreading

Lifecycle and States of a Thread

Main Thread

Synchronization

Inter-thread Communication & Java Concurrency

Explore More...

**File Handling**

File Class

File Permissions

Different ways of Reading a text file

Delete a File

Explore more...

**Garbage Collection**

Garbage Collection

Mark and Sweep

Explore more...

**Java Packages**

Packages

Java.io Package

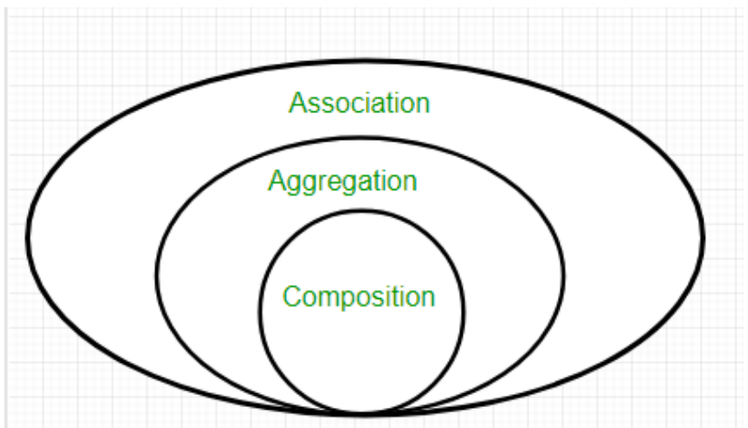
Java.lang package

| |
|---|
| Java.util Package |
| Networking |
| Socket Programming |
| URL class in Java |
| Reading from a URL |
| Inet Address Class |
| A Group Chat Application |
| Explore more... |

# Association, Composition and Aggregation in Java

**Association**

Association is relation between two separate classes which establishes through their Objects. Association can be one-to-one, one-to-many, many-to-one, many-to-many.
In Object-Oriented programming, an Object communicates to other Object to use functionality and services provided by that object. **Composition** and **Aggregation** are the two forms of association.




```
// Java program to illustrate the
// concept of Association
```

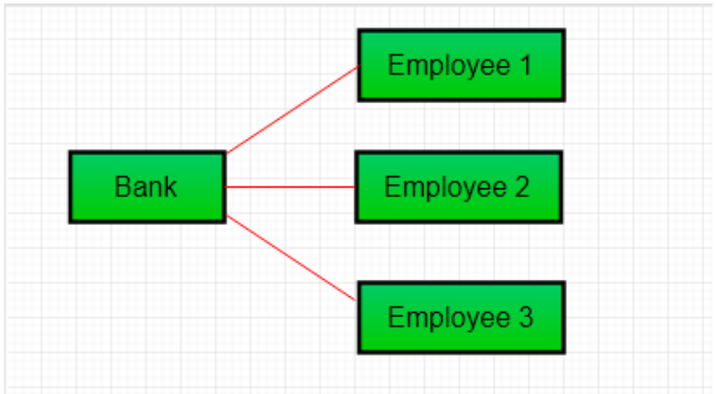
```java
import java.io.*;

// class bank
class Bank
{
    private String name;

    // bank name
    Bank(String name)
    {
        this.name = name;
    }

    public String getBankName()
    {
        return this.name;
    }
}

// employee class
class Employee
{
    private String name;

    // employee name
    Employee(String name)
    {
        this.name = name;
    }

    public String getEmployeeName()
    {
        return this.name;
    }
}

// Association between both the
// classes in main method
class Association
{
    public static void main (String[] args)
    {
        Bank bank = new Bank("Axis");
        Employee emp = new Employee("Neha");

        System.out.println(emp.getEmployeeName() +
               " is employee of " + bank.getBankName());
    }
}
```

Run on IDE

Output:

```
Neha is employee of Axis
```

In above example two separate classes Bank and Employee are associated through their Objects. Bank can have many employees, So it is a one-to-many relationship.




**Aggregation**

It is a special form of Association where:

- It represents **Has-A** relationship.
- It is a **unidirectional association** i.e. a one way relationship. For example, department can have students but vice versa is not possible and thus unidirectional in nature.
- In Aggregation, **both the entries can survive individually** which means ending one entity will not effect the other entity

```
// Java program to illustrate
//the concept of Aggregation.
import java.io.*;
import java.util.*;

// student class
class Student
{
    String name;
    int id ;
    String dept;

    Student(String name, int id, String dept)
```

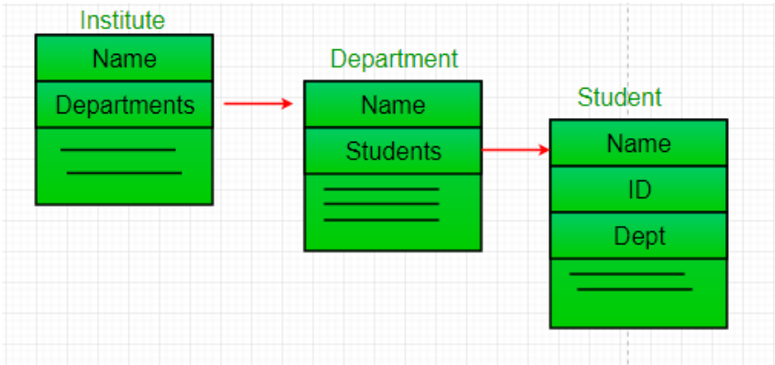
```
    {

        this.name = name;
        this.id = id;
        this.dept = dept;

    }
}

/* Department class contains list of student
Objects. It is associated with student
class through its Object(s). */
class Department
{

    String name;
    private List<Student> students;
    Department(String name, List<Student> students)
    {

        this.name = name;
        this.students = students;

    }

    public List<Student> getStudents()
    {
        return students;
    }
}

/* Institute class contains list of Department
Objects. It is asoociated with Department
class through its Object(s).*/
class Institute
{

    String instituteName;
    private List<Department> departments;

    Institute(String instituteName, List<Department> departments)
    {
        this.instituteName = instituteName;
        this.departments = departments;
    }

    // count total students of all departments
    // in a given institute
    public int getTotalStudentsInInstitute()
    {
        int noOfStudents = 0;
        List<Student> students;
        for(Department dept : departments)
```

```
        {
            students = dept.getStudents();
            for(Student s : students)
            {
                noOfStudents++;
            }
        }
        return noOfStudents;
    }

}

// main method
class GFG
{
    public static void main (String[] args)
    {
        Student s1 = new Student("Mia", 1, "CSE");
        Student s2 = new Student("Priya", 2, "CSE");
        Student s3 = new Student("John", 1, "EE");
        Student s4 = new Student("Rahul", 2, "EE");

        // making a List of
        // CSE Students.
        List <Student> cse_students = new ArrayList<Student>();
        cse_students.add(s1);
        cse_students.add(s2);

        // making a List of
        // EE Students
        List <Student> ee_students = new ArrayList<Student>();
        ee_students.add(s3);
        ee_students.add(s4);

        Department CSE = new Department("CSE", cse_students);
        Department EE = new Department("EE", ee_students);

        List <Department> departments = new ArrayList<Department>();
        departments.add(CSE);
        departments.add(EE);

        // creating an instance of Institute.
        Institute institute = new Institute("BITS", departments);

        System.out.print("Total students in institute: ");
        System.out.print(institute.getTotalStudentsInInstitute());
    }
}
```

Output:

```
Total students in institute: 4
```

In this example, there is an Institute which has no. of departments like CSE, EE. Every department has no. of students. So, we make a Institute class which has a reference to Object or no. of Objects (i.e. List of Objects) of the Department class. That means Institute class is associated with Department class through its Object(s). And Department class has also a reference to Object or Objects (i.e. List of Objects) of Student class means it is associated with Student class through its Object(s).
It represents a **Has-A** relationship.




**When do we use Aggregation ??**
Code reuse is best achieved by aggregation.

## Composition

Composition is a restricted form of Aggregation in which two entities are highly dependent on each other.

- It represents **part-of** relationship.
- In composition, both the entities are dependent on each other.
- When there is a composition between two entities, the composed object **cannot exist** without the other entity.

Lets take example of **Library**.

```
// Java program to illustrate
// the concept of Composition
```

```java
import java.io.*;
import java.util.*;

// class book
class Book
{

    public String title;
    public String author;

    Book(String title, String author)
    {

        this.title = title;
        this.author = author;
    }
}

// Libary class contains
// list of books.
class Library
{

    // reference to refer to list of books.
    private final List<Book> books;

    Library (List<Book> books)
    {
        this.books = books;
    }

    public List<Book> getTotalBooksInLibrary(){

       return books;
    }

}

// main method
class GFG
{
    public static void main (String[] args)
    {

        // Creating the Objects of Book class.
        Book b1 = new Book("EffectiveJ Java", "Joshua Bloch");
        Book b2 = new Book("Thinking in Java", "Bruce Eckel");
        Book b3 = new Book("Java: The Complete Reference", "Herbert Schildt");

        // Creating the list which contains the
        // no. of books.
        List<Book> books = new ArrayList<Book>();
        books.add(b1);
```

```
        books.add(b2);
        books.add(b3);

        Library library = new Library(books);

        List<Book> bks = library.getTotalBooksInLibrary();
        for(Book bk : bks){

            System.out.println("Title : " + bk.title + " and "
            +" Author : " + bk.author);
        }
    }
}
```

Run on IDE

Output

```
Title : EffectiveJ Java and  Author : Joshua Bloch
Title : Thinking in Java and  Author : Bruce Eckel
Title : Java: The Complete Reference and  Author : Herbert Schildt
```

In above example a library can have no. of **books** on same or different subjects. So, If Library gets destroyed then All books within that particular library will be destroyed. i.e. book can not exist without library. That's why it is composition.

**Aggregation vs Composition**

1. **Dependency:** Aggregation implies a relationship where the child **can exist independently** of the parent. For example, Bank and Employee, delete the Bank and the Employee still exist. whereas Composition implies a relationship where the child **cannot exist independent** of the parent. Example: Human and heart, heart don't exist separate to a Human
2. **Type of Relationship:** Aggregation relation is **"has-a"** and composition is **"part-of"** relation.
3. **Type of association:** Composition is a **strong** Association whereas Aggregation is a **weak** Association.

```
// Java program to illustrate the
// difference between Aggregation
// Composition.

import java.io.*;

// Engine class which will
// be used by car. so 'Car'
// class will have a field
// of Engine type.
class Engine
{
```

```
    // starting an engine.
    public void work()
    {

        System.out.println("Engine of car has been started ");

    }

}

// Engine class
final class Car
{

    // For a car to move,
    // it need to have a engine.
    private final Engine engine; // Composition
    //private Engine engine;     // Aggregation

    Car(Engine engine)
    {
        this.engine = engine;
    }

    // car start moving by starting engine
    public void move()
    {

        //if(engine != null)
        {
            engine.work();
            System.out.println("Car is moving ");
        }
    }
}

class GFG
{
    public static void main (String[] args)
    {

        // making an engine by creating
        // an instance of Engine class.
        Engine engine = new Engine();

        // Making a car with engine.
        // so we are passing a engine
        // instance as an argument while
        // creating instace of Car.
        Car car = new Car(engine);
        car.move();

    }
```

```
}
```

Run on IDE

Output:

```
Engine of car has been started
Car is moving
```

In case of aggregation, the Car also performs its functions through an Engine. but the Engine is not always an internal part of the Car. An engine can be swapped out or even can be removed from the car. That' why we make The Engine type field non-final.

This article is contributed by **Nitsdheerendra**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

**Practice Tags :** Java

**Article Tags :** Java java-inheritance

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

## Recommended Posts:

Access and Non Access Modifiers in Java
Access Modifiers in Java
Dynamic Method Dispatch or Runtime Polymorphism in Java
Overloading in Java
Using final with Inheritance in Java

Java AWT | Color Class
HashSet iterator() Method in Java
HashSet clone() Method in Java
HashSet clear() Method in Java
HashSet contains() Method in Java

(Login to Rate)

2.9 Average Difficulty : **2.9/5.0**
Based on **25** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!



@geeksforgeeks, Some rights reserved