

# LAB MANUAL

CS1421: Object Oriented Programming

LAB 10

**Instructors**

Aman Farooq

Lab 10

Association, Aggregation And Composition

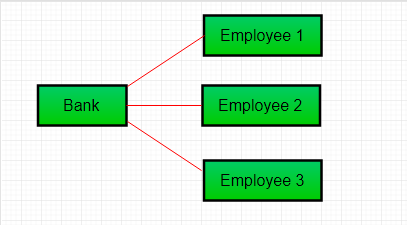
**Objective**

* Understand the concept of composition , aggregation and association

**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.  
 [](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Associatn.png)

It represents a relationship between two or more objects where all objects have their own lifecycle and there is no owner. The name of an association specifies the nature of relationship between objects. This is represented by a solid line.



**Example:**

// Java program to illustrate the

// concept of Association

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("Ali");

        System.out.println(emp.getEmployeeName() +

               " is employee of " + bank.getBankName());

    }

}

Output:

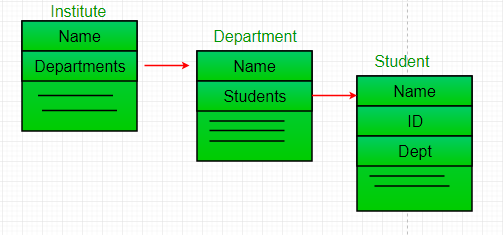
Ali is employee of Axis

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 affect 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

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**.

**Example:**

|  |
| --- |
| // Java program to illustrate  // the concept of Composition  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);          }      }  } |

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.

Another Example:

// 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();

    }

}

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.

**MAP:**

A map is a container object that stores a collection of key/value pairs. It enables fast retrieval, deletion, and updating of the pair through the key. A map stores the values along with the keys. The keys are like indexes. In List, the indexes are integers. In Map, the keys can be any objects. A map cannot contain duplicate keys. Each key maps to one value. A key and its corresponding value form an entry stored in a map.

You can create a map using one of its three concrete classes: HashMap, LinkedHashMap, or TreeMap.

import java.util.\*;

public class TestMap {

public static void main(String[] args) {

// Create a HashMap

Map<String, Integer> hashMap = new HashMap<>();

hashMap.put("Smith", 30);

hashMap.put("Anderson", 31);

hashMap.put("Lewis", 29);

hashMap.put("Cook", 29);

System.out.println("Display entries in HashMap");

System.out.println(hashMap + "\n");

}

}

**Practice Tasks**

1. There is a hospital UMH that needs to develop a system to automate its operations. **Hospital** has several departments (**Cardiology, Emergency, Critical Care, Neurology etc and a pharmacy).** There are many **doctors** in hospital. Multiple doctors are assigned to each department according to current needs, one doctor can have many patients assigned. Patients admitted to hospital get assigned a department according to disease and a doctor from that department. Doctor can recommend medicine to patient form hospital **Pharmacy**.

Develop a system for UMH hospital. Create departments for hospital. Add doctors in each department. Assign patients to doctors. Make Doctor recommend medicine to patient from pharmacy.

1. Identify Classes
2. Identify relationships between classes
3. Develop a Program to automate this system.

In your Program

* Hospital has 3 departments (any)
* Each department has three doctors assigned
* Each doctor has two patients to treat

After designing your system, answer the following questions with reasons.

1. If hospital decides to close its operations on permanent basis what would be the situation then?
2. Will departments be there working?
3. Will doctors exist?
4. If hospital management decides to close one of its department?
5. Will doctors be still there in hospital?
6. Will patients of that department be there?
7. If a patient is discharged from hospital.
8. Will his doctor, pharmacy or department get affected?
9. If hospital management decides to close its pharmacy?
10. Will doctors get affected?

1. In your OOP class, students are registered by S.R No and name. But assignment, quiz and Exams marks are being uploaded with SR. No and Marks. Design a system in which Student can select whether they want to view their Quiz, Assignment or Exam marks. Student will input their name to view any of this. Get Student’s S.R no from OOP Students’ list and look for their marks in desired list (quiz, assignment or Exam). Show desired output to student.