Association

**Association**

**Association** is a relationship between two objects. Here objects might not be completely dependent on each other that means there is no ownership and both the objects have their own life cycle.

Association can be one-to-one, one-to-many, many-to-one, and many-to-many.

**In association** **objects have an independent lifecycle** and where there is **no ownership between the objects**.

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.

**public** **class** Student {

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

Teacher teacher = **new** Teacher();

teacher.course();

System.***out***.println("DONE");

}

}

**public** **class** Teacher {

**public** **void** course(){

System.***out***.println("J2EE COURSE");

}

}

Let’s take an example of a teacher and student.

Multiple students can associate with a single teacher, and a single student can associate with multiple teachers, but both have their own lifecycles (both can be create and delete independently).

So when a teacher leaves the school, we don’t need to delete any students, and when a student leaves the school, we don’t need to delete any teachers.



**class** Bank {

**private** String name;

// bank name

Bank(String name) {

**this**.name = name;

}

**public** String getBankName() {

**return** **this**.name;

}

}

**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());

}

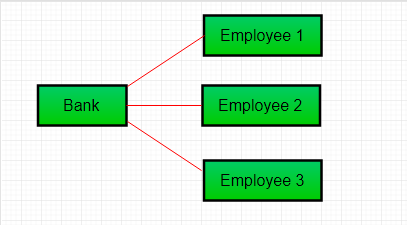
}

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:

* A has-A relationship is represented here.
* It is a one-way relationship, i.e. a unidirectional relationship.
* Ending one entity won’t affect another, both can be present independently.

**public** **class** Address {

String city, state, country;

**public** Address(String city, String state, String country) {

**this**.city = city;

**this**.state = state;

**this**.country = country;

}

}

**public** **class** Emp {

**int** id;

String name;

Address address;

**public** Emp(**int** id, String name, Address address) {

**this**.id = id;

**this**.name = name;

**this**.address = address;

}

**void** display() {

System.***out***.println(id + " " + name);

System.***out***.println(

address.city + " " + address.state + " " + address.country);

}

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

Address address1 = **new** Address ("Ranchi", "Jharkhand", "india");

Address address2 = **new** Address ("Gaya", "Bihar", "india");

Emp e = **new** Emp(111, "Amit", address1);

Emp e2 = **new** Emp(112, "Rakesh", address2);

e.display();

e2.display ();

}

}

Output

111 Amit

Ranchi Jharkhand india

112 Rakesh

Gaya Bihar india

Employee has an object of Address; address object contains its own information such as city, state, country etc. In such case relationship is Employee HAS-A address.

* **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.

**public** **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.

**public** **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;

}

}

**public** **class** Composition {

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

// Creating the Objects of Book class.

Book b1 = **new** Book("Java", "Jamesh Gosline");

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 : Java and Author : Jamesh Gosline

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 cannot exist without library. That’s why it is composition.

**Generalization basic points**

* Type/Subtype Relationship
* "IS A " Relationship
* One class has many subclasses
* General class can have many Special classes as  subclasses
* General names can be used to Address special names
* Top to bottom hierarchy goes towards specialization
* Bottom to top Hierarchy goes towards generalization

The process of extracting common characteristics from two or more classes and combining them into a generalized super class is called Generalization.

The common characteristics can be attributes or methods.

Generalization is represented by a triangle followed by a line.

https://dotnettricksweb.blob.core.windows.net/img/oops/generalizationarrow.png

Specialization is the reverse process of Generalization means creating new sub classes from an existing class.

Let’s take an example of Bank Account;

A Bank Account is of two types – Current Account and Saving Account.

Current Account and Saving Account inherits the common/ generalized properties like Account Number, Account Balance etc. from a Bank Account and also have their own specialized properties like interest rate etc.

