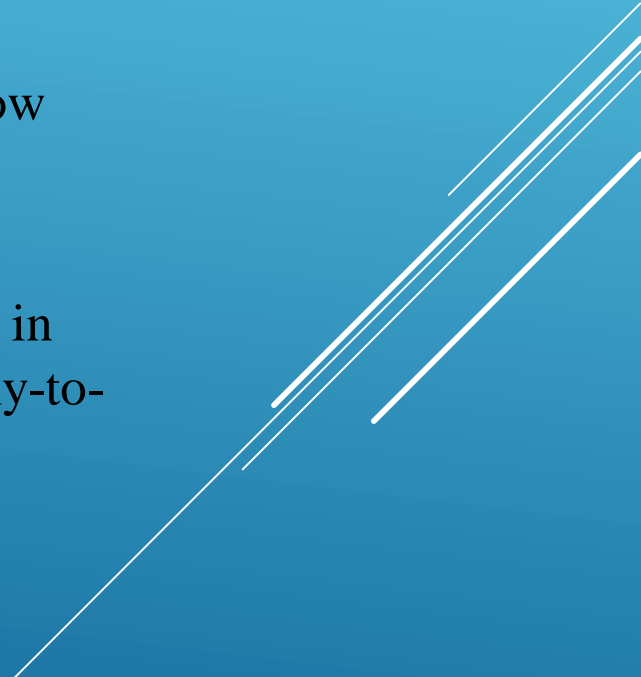


JAVA ASSOCIATION AND THEIR TYPES

-K.L.Madhavi

ASSOCIATION

- ▶ Association is the cardinal concept in object-oriented programming that describes the relationship between two independent classes.
 - ▶ It indicates how objects of classes communicate with each other and how they use the functionality and services provided by that communicated object.
 - ▶ Association may be either unidirectional or bidirectional and may exist in several forms, such as one-to-one, one-to-many, many-to-one, and many-to-many.
- 
- A series of three parallel white diagonal lines are positioned in the bottom right corner of the slide, extending from the right edge towards the center.

```
class Owner {
    String name;
    Owner(String name) {
        this.name = name;
    }
}

class Car {
    String name;
    Student(String name) {
        this.name = name;
    }
}

public class DrivingSchool {
    public static void main(String[] args) {
        Owner owner = new Owner("abdul");
        Car car = new car("Maruti swift car");
        System.out.println(owner.name + " drives " + Car.name);
    }
}
```

In this case, there's an association between Owner and Car since a Owner drives a car, but they are independent entities.

TYPES OF RELATIONS IN ASSOCIATION

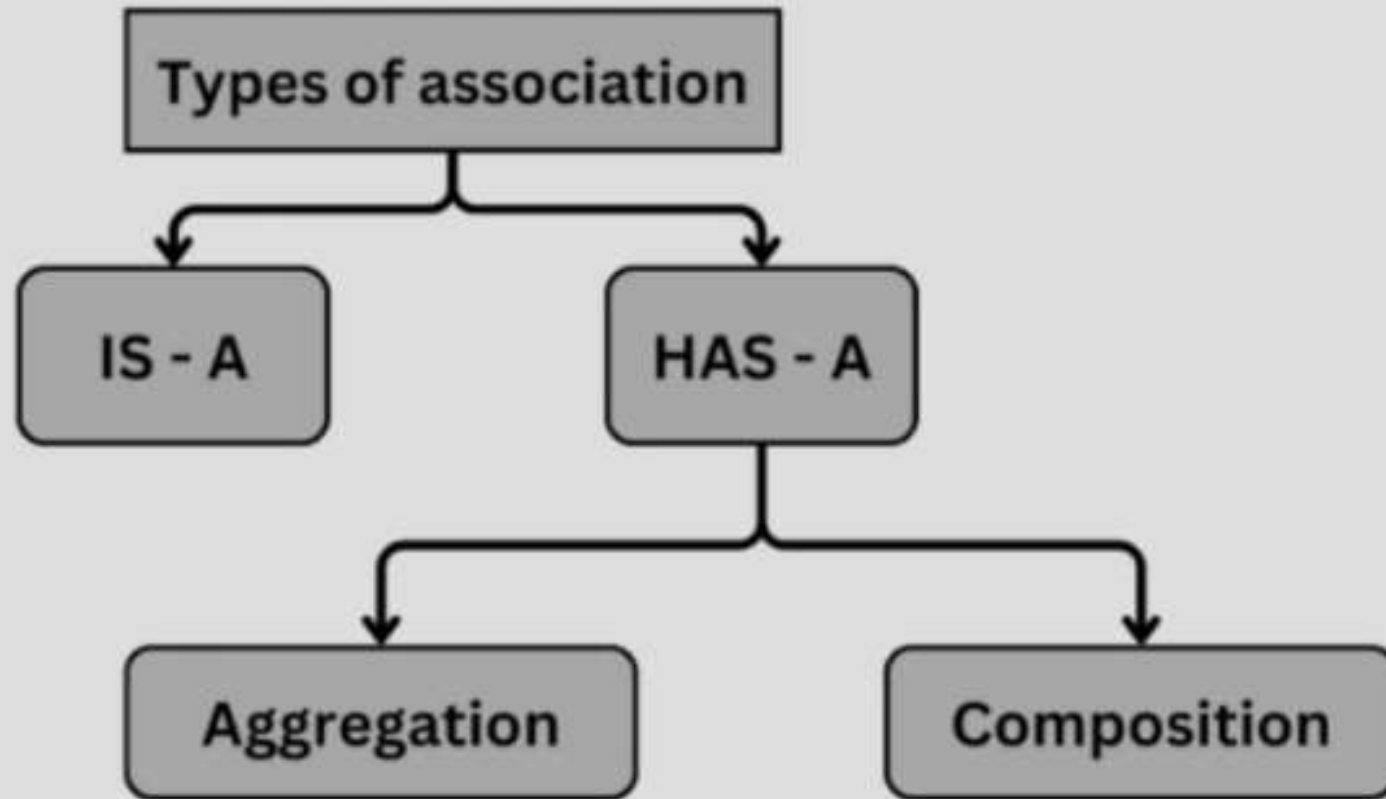
1. Unidirectional Association:

This association is the one in which one class is aware and associated with another class; the reverse is not true. For example, the Student class can be associated with the LibraryCard class, for the association where the student has a library card; a LibraryCard does not need to know about a Student.

2. Bidirectional Association:

In this type of association, the classes are aware of each other and interact with one another. For example, a Teacher class and a Classroom class may be associated bidirectionally; there would be a teacher assigned to a classroom, and a classroom would know to which teacher it is assigned.

TYPES OF ASSOCIATION




AGGREGATION

- ▶ In Java, Aggregation is a type of relationship where one class is used by another class but does not own it. In other words, the class that does not own it still has access to all of its data and behavior.
- ▶ It's a weaker association than composition because the lifetime of the class being used is not dependent on the lifetime of the class using it.
- ▶ A classic example of aggregation is the relationship between a college and a student. A student belongs to one college with single regno, while a college contain no of students. Thus, the relationship between a college and a student is an example of aggregation.

```
class College {  
    String name;  
    College(String name) {  
        this.name = name;  
    }  
}  
  
class Student {  
    String regno;  
    Student(String regno) {  
        this.regno= regno;  
    }  
}  
  
public class AggregationExample {  
    public static void main(String[] args) {  
        College college= new College("Pragati College");  
        Student student = new Student("20A31A4212");  
        System.out.println(student.regno + " is available at " +  
college.name);  
    }  
}
```

In this example, College aggregates Student, meaning that Student exist in the College independently. Here if Student is not there College will Exist.

COMPOSITION

- ▶ Composition is a type of relationship between two objects in which one class owns. In other words, the class that owns the other class is considered to be composed of the other class. another class and can access all its data and behavior.
 - ▶ Composition is a powerful concept in Java, as it allows for the creation of complex objects from simpler ones. It also allows for the reuse of code, as the same object can be used in multiple classes.
 - ▶ Composition helps to reduce code size by increasing the code reusability; it also provides better control over the objects at run time.
- 
- Several white lines of varying lengths and slopes are positioned in the bottom right corner of the slide, creating a modern, abstract graphic element.


```
class Battery{
    String type;
    Battery(String type) {
        this.type = type;
    }
}

Class Clock {
    Battery battery;
    Clock(String batteryType) {
        this.battery = new Battery(batteryType);
    }
}

public class CompositionExample {
    public static void main(String[] args) {
        Clock clock = new Clock("Duracell battery");
        System.out.println("Clock has an battery of type: " +
clock.battery.type);
    }
}
```

In this case, Clock is composed of an battery. Here Clock and battery both have strong relationship between them. Here if battery is destroyed clock never work.