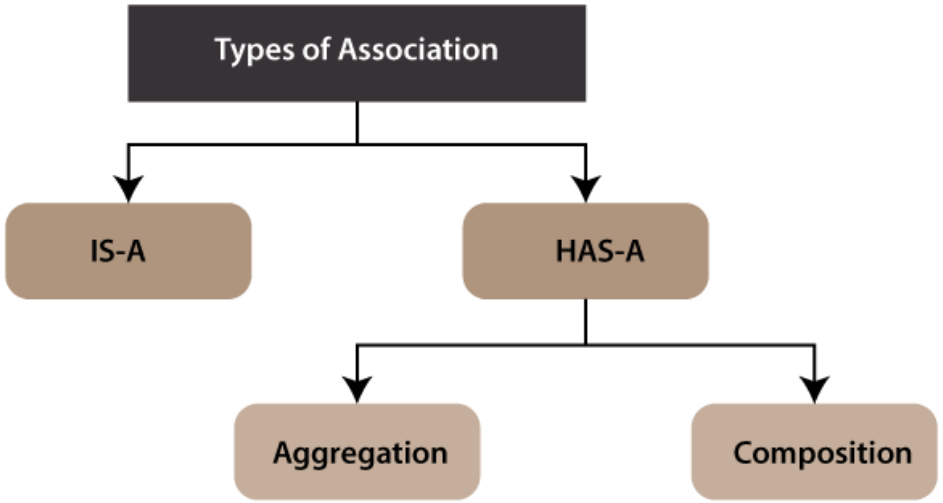
Lecture – 10

Association, Aggregation
and Composition

Association

- Interaction of different objects in OO model (or in problem domain) is known as **association**
- In object-oriented model, objects interact with each other to perform some useful work
  - Modeling these objects (entities) is done using the association
- This association can be represented with a line along an arrowhead () or without arrowhead

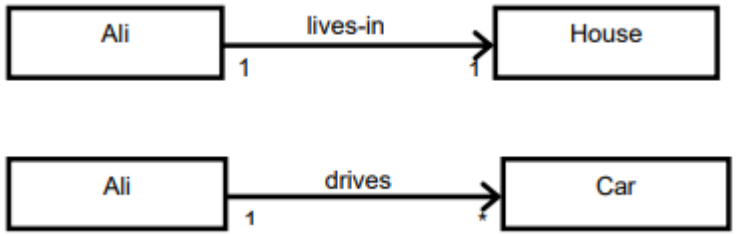
Types of Association



- **Class Association – Inheritance (IS-A)**
- **Object Association**
  - It is the interaction of stand-alone objects of one class with other objects of anther class.
  - **Simple Association or Association**
  - **Composition**
  - **Aggregation**

Types of Association

- Simple Association
  - The two interacting objects have no intrinsic relationship with other object
  - It is the weakest link between objects.
  - It is a reference by which one object can interact with some other object
  - Example:
    - Customer gets cash from cashier
    - Employee works for a company
    - Ali lives in a house
    - Ali drives a car

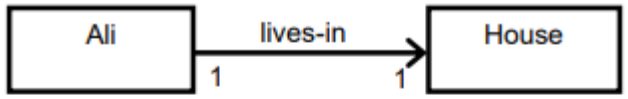


Types of Association

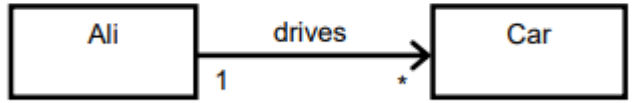
- Simple association can be categorized in two ways

With respect to direction (navigation)

One-Way Association

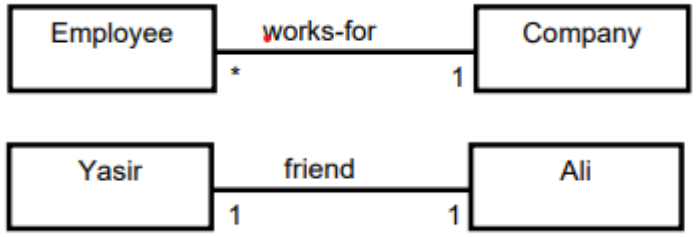


Ali lives in a House



Ali drives his Car

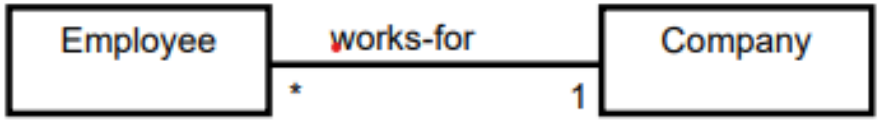
Two-Way Association



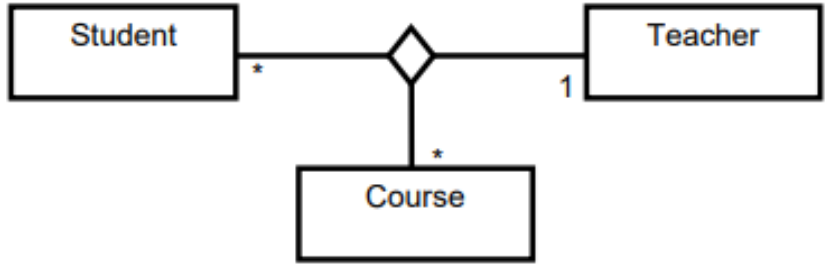
With respect to number of objects (cardinality)

Binary Association

It associates objects of exactly two classes; it is denoted by a line, or an arrow between the associated objects.

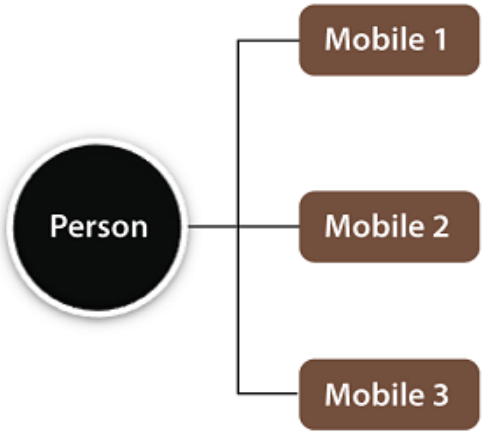


Ternary Association



Association Relationships Types

- One-to-One
  - One instance of a class is related to one instance of another class
  - **Example:** A person can have only one passport.
- One-to-Many
  - One instance of a class is related to many instances of another class
  - **Example:** A College can have many students
- Many-to-Many
  - Many instances of one class are related to many instances of another class.
  - **Example:** A single student can associate with multiple teachers, and multiple students can also be associated with a single teacher.



Aggregation(Has-A)

- The relationship between two classes called whole/part
- It is unidirectional relationship.
- Implementation:
  - One class has a reference to another class, but the second class can exist independently of the first class.
- Benefit: Code Reusability
- Examples
  - Student Has-A Address (Has-A relationship between student and address)
  - Staff Has-A Address (Has-A relationship between staff and address)
  - Department Has-A Teacher (Has-A relationship between department and teacher)

Aggregation(Has-A)

- Example:
  - Department(A) Has-A Teacher(B)
  - A owns B
  - The lifetime of object B does not depend on lifetime of object A
  - If there is no department, it does not mean teacher doesn’t exist.

Example

- Consider two classes Student class and Address class. Every student has an address so the relationship between student and address is a Has-A relationship. But if you consider its vice versa then it would not make any sense as an Address doesn't need to have a Student necessarily.

```
public class Address{
  private int streetNum;
  private String city;
  private String state;
  private String country;
  Address(int street, String c, String st, String coun){
     this.streetNum=street;
     this.city =c;
     this.state = st;
     this.country = coun;
  }
  public int getStreetNum(){
    return this.streetNum;
  }
  public String getCity(){
    return this.city;
  }
  public String getState(){
    return this.state;
  }
  public String getCountry(){
    return this.country;
  }
}
```

```
public class Student{
  private int rollNum;
  private String studentName;
  //Creating HAS-A relationship with Address class
  private Address studentAddr;
  Student(int roll, String name, Address addr){
     this.rollNum=roll;
     this.studentName=name;
     this.studentAddr = addr;
  }
  public int getRollNum(){
    return this.rollNum;
  }
  public String getStudentName(){
    return studentName;
  }
  public Address getAddress(){
    return this.studentAddr;
  }
   public int getStreetNum(){
    return studentAddr.getStreetNum();
  }
```

```
public String getCity(){
    return studentAddr.getCity();
  }
  public String getState(){
    return studentAddr.getState();
  }
  public String getCountry(){
    return studentAddr.getCountry();
  }
}
```

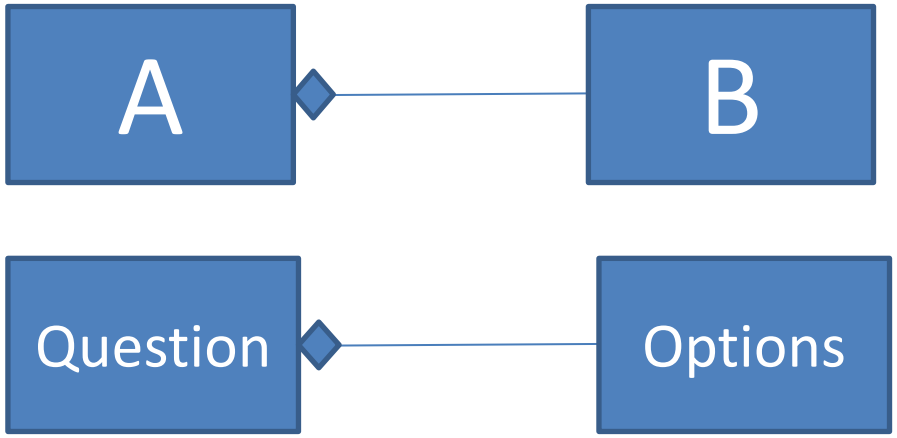
```
public class StudentAddressTest{
public static void main(String args[]){
    Address addr = new Address(55, "Islamabad", "Fedral", "Pakistan");
    Student ahmad = new Student(123, "Ahmad", addr);
    System.out.println(ahmad.getRollNum());
    System.out.println(ahmad.getStudentName());
    System.out.println(ahmad.getAddress().getStreetNum());
    System.out.println(ahmad.getCity());
    System.out.println(ahmad.getState());
    System.out.println(ahmad.getCountry());
  }
}
```

Composition

- Composition is a specialized form of aggregation.
- In composition, if the parent object is destroyed, then the child objects will not exist.
- Composition is a strong type of aggregation and is sometimes referred to as a “death” relationship
- Example:
  - House Has-A Room or a house may be composed of one or more rooms
  - If the house is destroyed, then all of the rooms that are part of the house are also destroyed

Composition

- In composition the life cycle of the part or child is controlled by the whole or parent that owns it
- Example
  - Question(A) Has-A Options(B)




```
public class House
{
  private Room room;
  public House()
  {
     room = new Room();
  }
}
```

```
public class Question{
    private String questionText;
    private Option option1, option2, option3;
    public Question(String questionText) {
        this.questionText = questionText;
        option1 = new Option("Option 1", false);
        option2 = new Option("Option 2", true);
        option3 = new Option("Option 3", false);
    }
    public String getQuestionText() {
        return questionText;
    }
    public String getOption1() {
        return option1.getOptionText();
    }
    public String getOption2() {
        return option2.getOptionText();
    }
    public String getOption3() {
        return option3.getOptionText();
    }
}
```

```
public class Option{
    private String optionText;
    private boolean isCorrect;

    public Option(String optionText, boolean isCorrect) {
        this.optionText = optionText;
        this.isCorrect = isCorrect;
    }

    public String getOptionText() {
        return optionText;
    }

    public boolean isCorrect() {
        return isCorrect;
    }

}
```

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


      Question question = new Question("Which option is correct?");

      System.out.println(question.getQuestionText());
      System.out.println(question.getOption1());
      System.out.println(question.getOption2());
      System.out.println(question.getOption3());
   }
}
```

```
Which option is correct?
Option 1
Option 2
Option 3
```