Object-oriented programming - relations

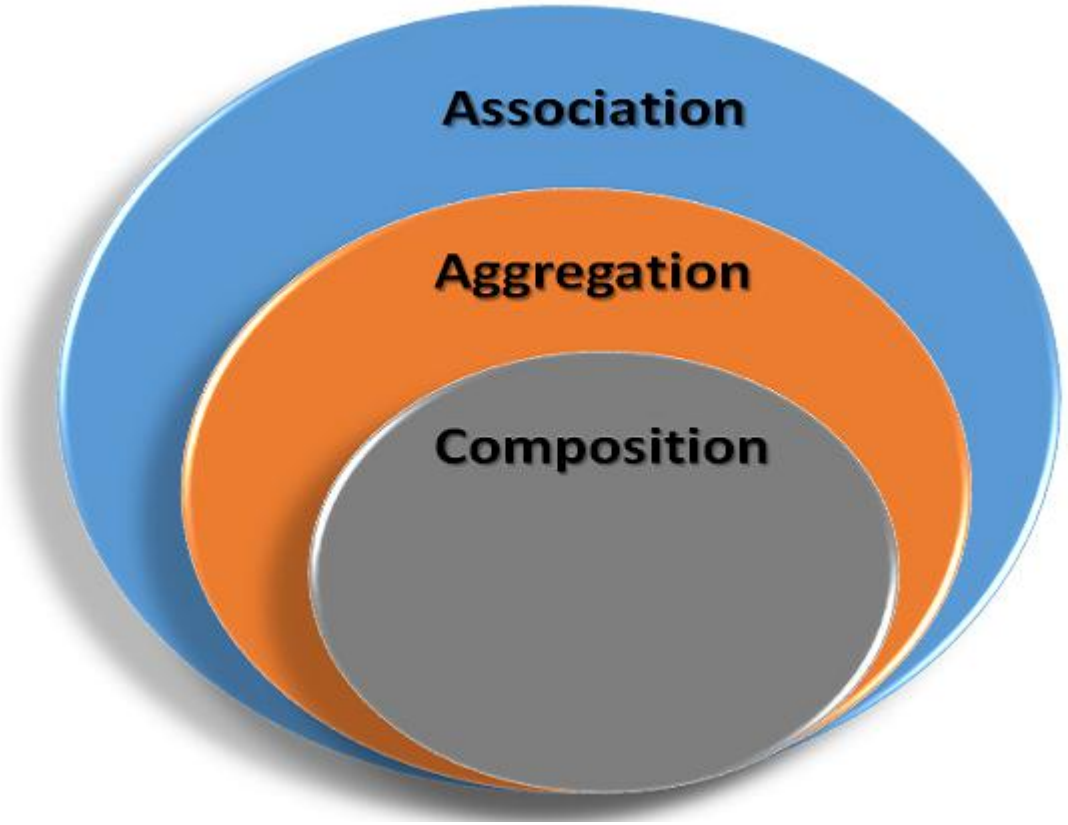
Rafał Witkowski

2021

Relations

- Code reuse is one of the many benefits of OOP (object-oriented programming). Reusability is feasible because of the various types of relationships that can be implemented among classes. This presentation will demonstrate the types of relationships (from weak to strong) using Java code samples and the symbols in the UML (unified modeling language) class diagram.

Relations



Dependency (Using)

- Is a relationship when objects of a class work briefly with objects of another class. Normally, multiplicity doesn’t make sense on a dependency

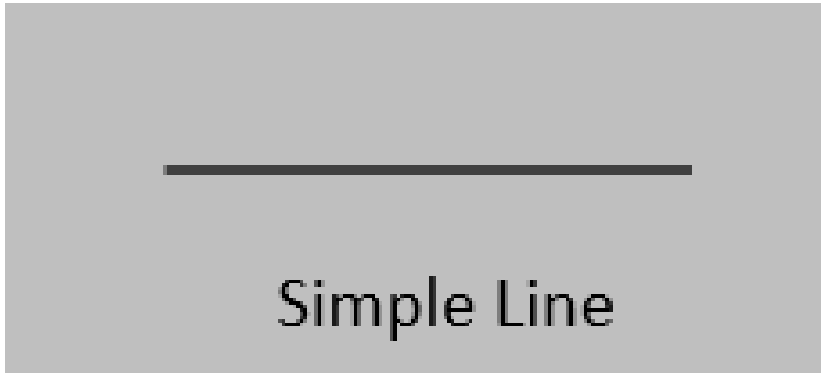
Dependency - Java

```
class Move {
   public void Roll() { ... }
}

class Player {
   public void PerformAction(Move move) {
      move.Roll();
   }
}
```

Association

- This relationship transpires when objects of one class know the objects of another class. The relationship can be one to one, one to many, many to one, or many to many. Moreover, objects can be created or deleted independently.



Association – java

```
public static void main (String[] args) {
   Doctor doctorObj = new Doctor("Rick");
   Patient patientObj = new Patient("Morty");
   System.out.println(patientObj.getPatientName() +
        " is a patient of " + doctorObj.getDoctorName());
}
```

Aggregation

- Is a "has-a" type relationship and is a one-way form of association. This relationship exists when a class owns but shares a reference to objects of another class.

Aggregation – java

```
class Address{
//code here
}

class StudentClass{
   private Address studentAddress;
//code here
}

Public StudentClass(Address adres)
{
  studentAddress = adres;
}
```

Composition

- Is a "part-of" type of relationship, and is a strong type of association. In other words, composition happens when a class owns & contains objects of another class

Composition – java

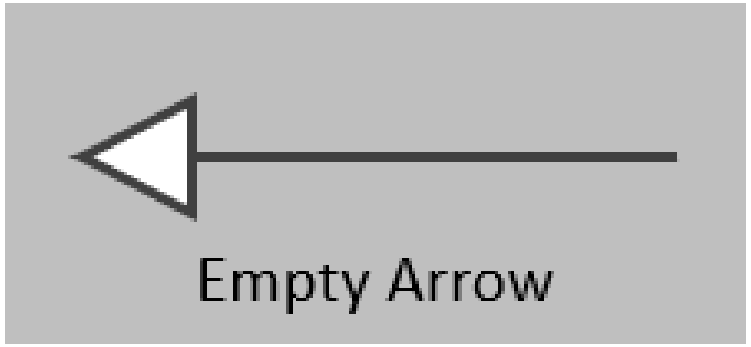
```
class Room {
//code here
}

class House {
   private Room room;
//code here
}

Public House()
{
  room = new Room();
}
```

Inheritance

- Is an "is-a" type of relationship. It's a mechanism that permits a class to acquire the properties and behavior of an existing class (or sometimes more classes, if the language allows multiple inheritance).



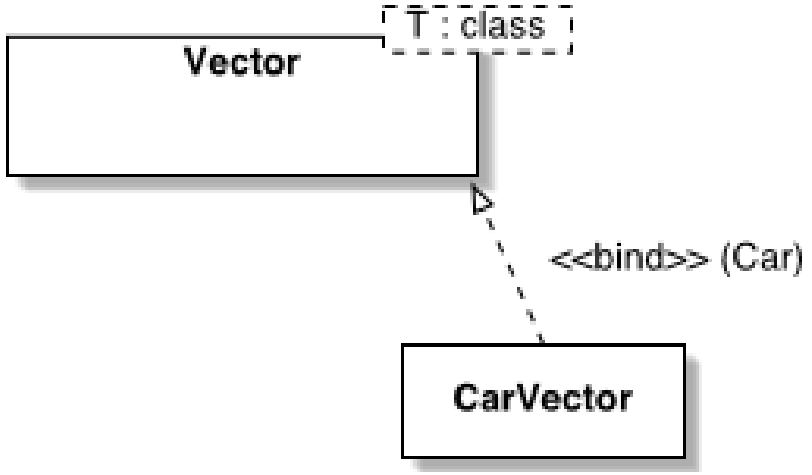
Interihence – java

```
class Vehicle {
//code here
}

class Car extends Vehicle {
//code here
}
```

Instantiation (generics)

- A generic class arises from a situation where the behavior of a class can be abstracted into something which is relevant to more than one type of state. The goal is to share the definition (in the class) of some behavior (i.e. the methods) across different data types.



Polymorphism

- For OO languages polymorphism is tied up with substitutability. We design methods and we write client code that can operate on a set of types. What is common about these types is that they are substitutable for each other.
- The idea that the code that is executed as the result of a message being sent depends on the class of the object that receives the message. Objects of different classes can react differently to being sent the same method in a message.