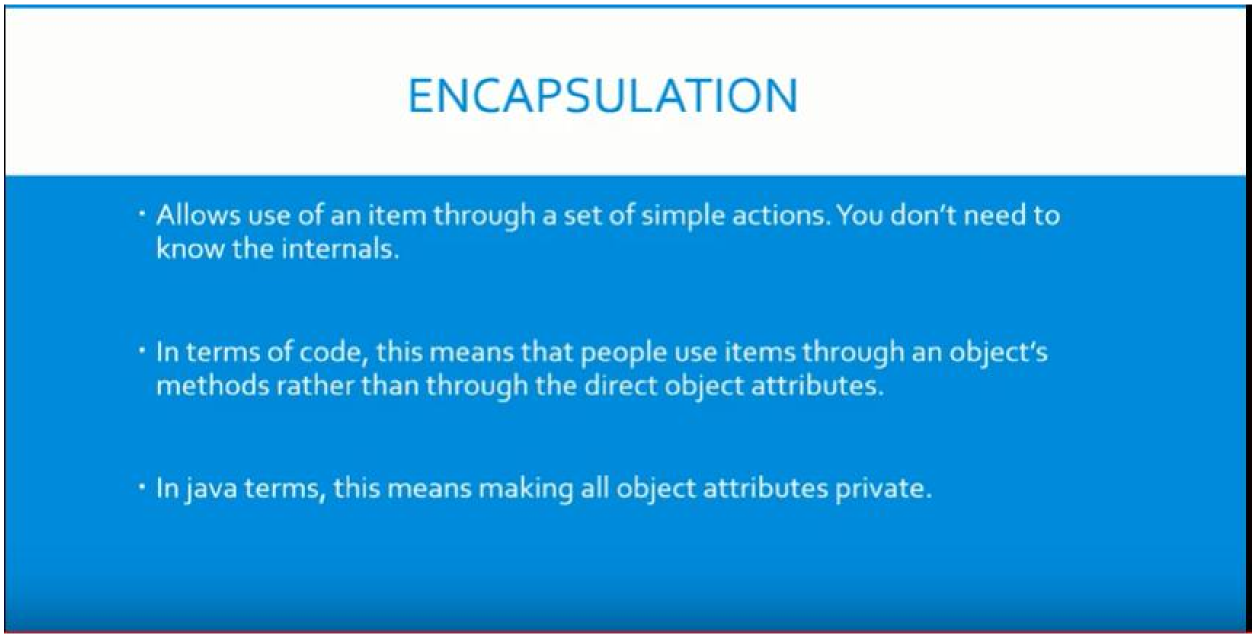
1. What are the 4 principles of OOP?

- Polymorphism.
- Abstraction.
- Encapsulation.
- Inheritance.

2. What is Encapsulation?

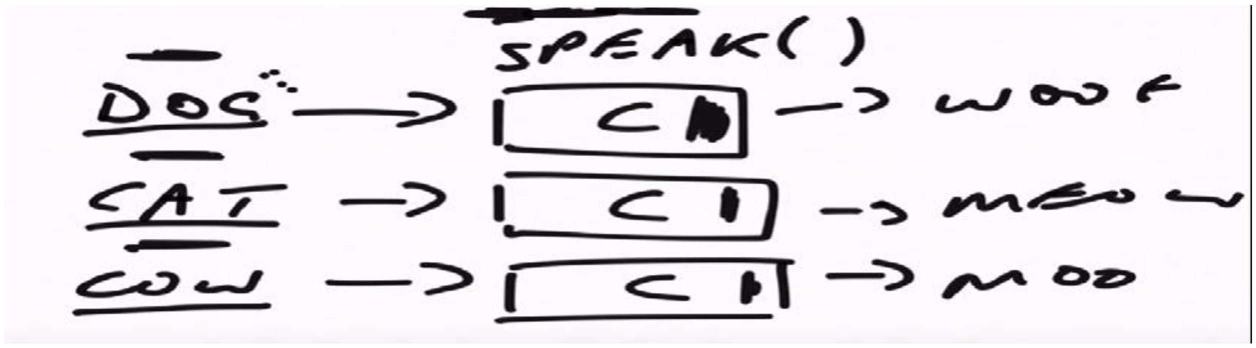


ENCAPSULATION

- Allows use of an item through a set of simple actions. You don't need to know the internals.
- In terms of code, this means that people use items through an object's methods rather than through the direct object attributes.
- In java terms, this means making all object attributes private.

3. What is Polymorphism? → Many Forms

- So polymorphism is the ability (in programming) to present the same interface for differing underlying forms (data types).
- **Polymorphism** in Java has two types: Compile time **polymorphism** (static binding) and Runtime **polymorphism** (dynamic binding). Method **overloading** is an example of static **polymorphism**, while method overriding is an example of dynamic **polymorphism**.



4. What is Abstraction?

Abstraction in the real world

I'm a coffee addict. So, when I wake up in the morning, I go into my kitchen, switch on the coffee machine and make coffee. Making coffee with a coffee machine is a good example of abstraction.

You need to know how to use your coffee machine to make coffee. You need to provide water and coffee beans, switch it on and select the kind of coffee you want to get. The thing you don't need to know is how the coffee machine is working internally to brew a fresh cup of delicious coffee.

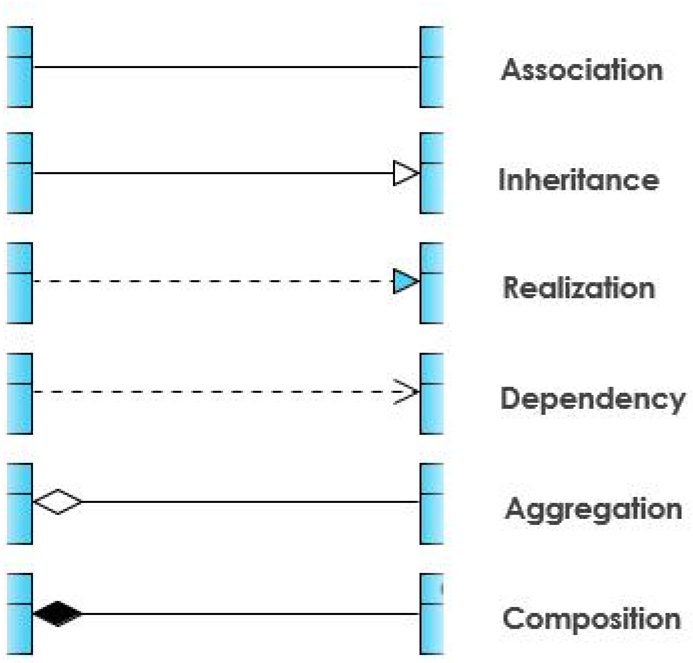
Abstraction in OOP

Objects in an OOP language provide an abstraction that hides the internal implementation details. Similar to the coffee machine in your kitchen, you just need to know which methods of the object are available to call and which input parameters are needed to trigger a specific operation.

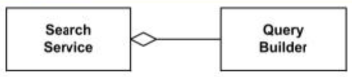
```java
public class CoffeeMachine {
    private Map<CoffeeSelection, CoffeeBean> beans;

    public CoffeeMachine(Map<CoffeeSelection, CoffeeBean> beans) {
         this.beans = beans
    }

    public Coffee brewCoffee(CoffeeSelection selection) throws CoffeeException {
        Coffee coffee = new Coffee();
        System.out.println("Making coffee ...");
        return coffee;
    }
}
```

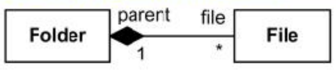
```
public enum CoffeeSelection {
    FILTER_COFFEE, ESPRESSO, CAPPUCCINO;
}
```

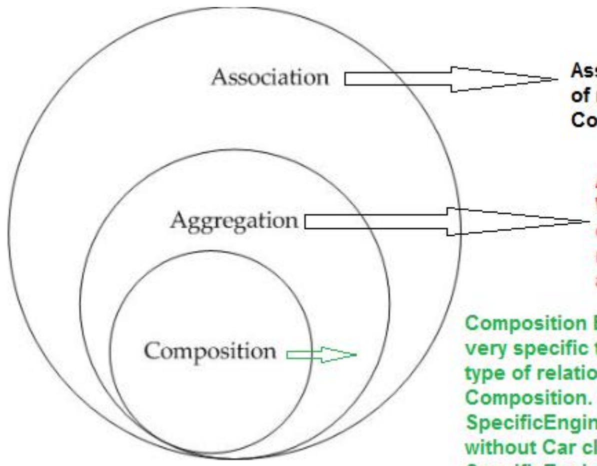
5. UML Notation:



Note: Inheritance (or Generalization): Both are same

Aggregation	
 ***Search Service*** *has a* ***Query Builder*** *using shared* ***aggregation***	**Aggregation** (aka **shared aggregation**) is shown as binary association decorated with a **hollow diamond** as a terminal adornment at the aggregate end of the association line.

Composite Aggregation (Composition)	
 *Folder could contain many files, while each File has exactly one Folder parent. If Folder is deleted, all contained Files are deleted as well.*	**Composite aggregation (aka composition)** is a "strong" form of **aggregation**. **Composition** is depicted as binary association decorated with a **filled black diamond** at the aggregate (composite) end.



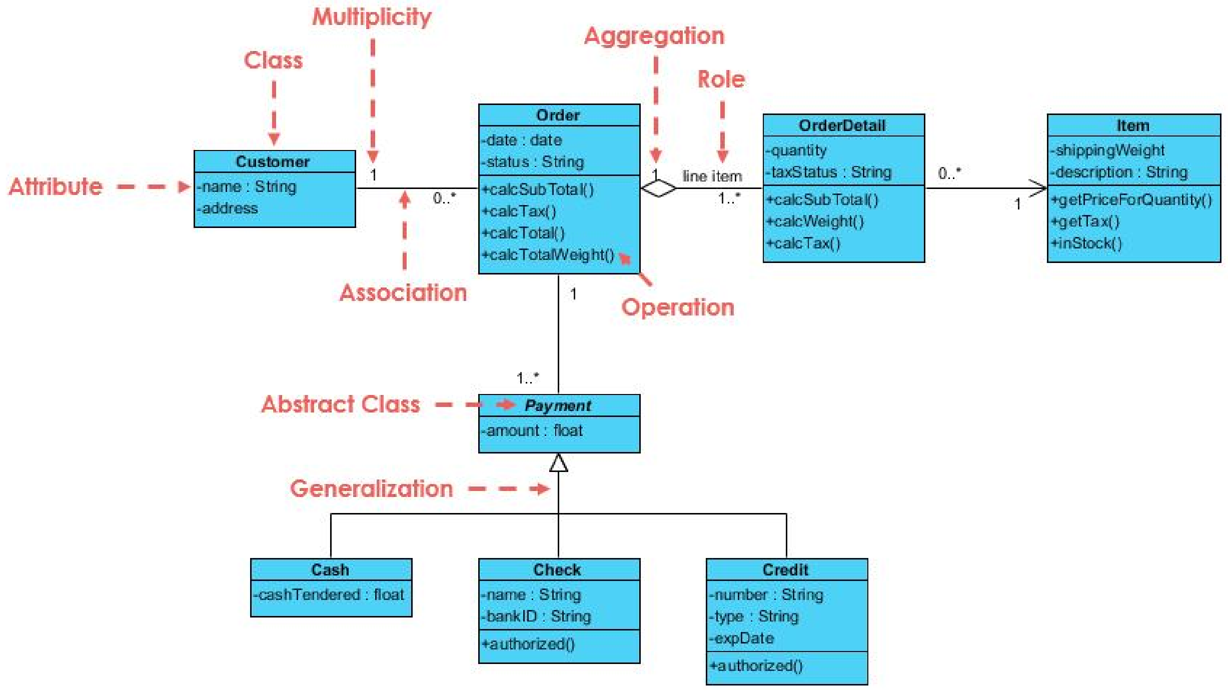
Association is generalized concept of relations. It includes both Composition and Aggregation.

Aggregation Example : Now consider class Car and class Wheel. Car needs a Wheel object to function. Meaning Car object own Wheel object but we cannot say Wheel object has no significance without Car Object. It can very well be used in a Bike, Truck or different Cars Object.

Composition Example : Consider example of a Car and an engine that is very specific to that car (meaning it cannot be used in any other car). This type of relation ship between Car and SpecificEngine class is called Composition. Object of Car class cannot exist without object of SpecificEngine class and object of SpecificEngine has no significance without Car class. To put in simple words Car class solely "owns" the SpecificEngine class.

Note: In both aggregation and composition object of one class "owns" object of another class.

Class Diagram Example: Order System


Multiplicity
Class
Aggregation
Role
Attribute
Customer
-name : String
-address
1
0..*
Order
-date : date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
1
line item
1..*
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
0..*
1
Item
-shippingWeight
-description : String
+getPriceForQuantity()
+getTax()
+inStock()
Association
Operation
1
1..*
Abstract Class
Payment
-amount : float
Generalization
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
Credit
-number : String
-type : String
-expDate
+authorized()