Java™

Inheritance

Another principle of OOPs

Inheritance & it's types

- The Inheritance is a process of obtaining the data members and methods from one class to another class, plus can have its own is known as inheritance.
- Inheritance - IS-A relationship between a superclass and its subclasses.
- The class which property is used to create new class is known as **Base/Parent/Super** class and the class which is created with the help of super class is known as **Derive/Child/Sub** class.

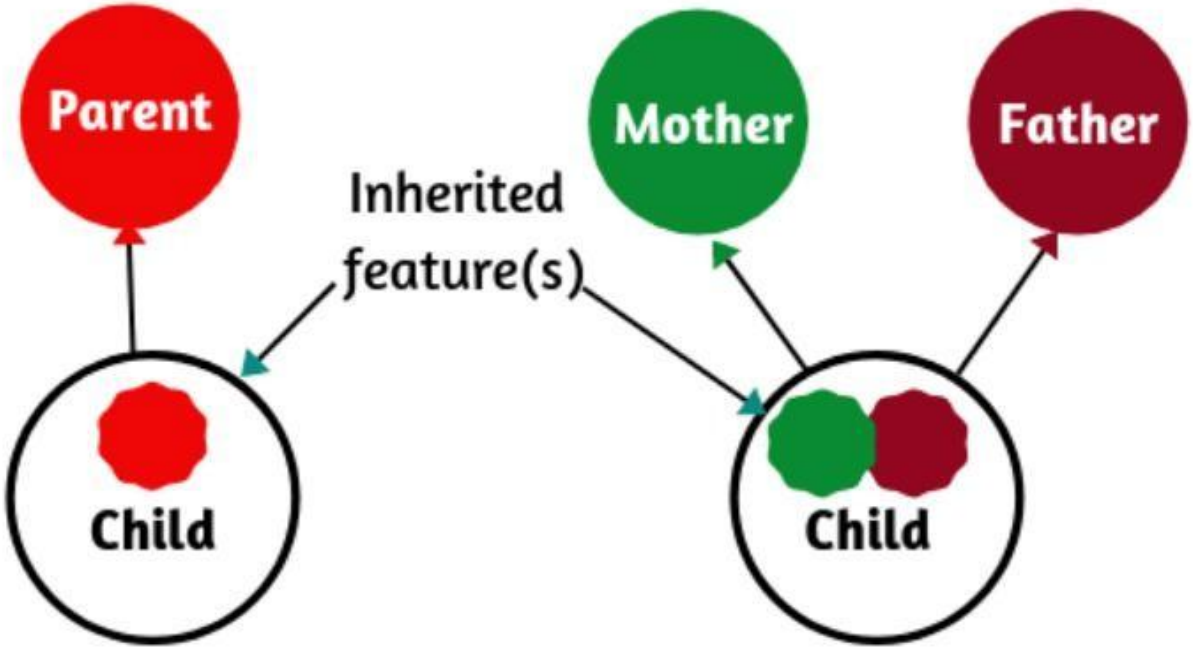
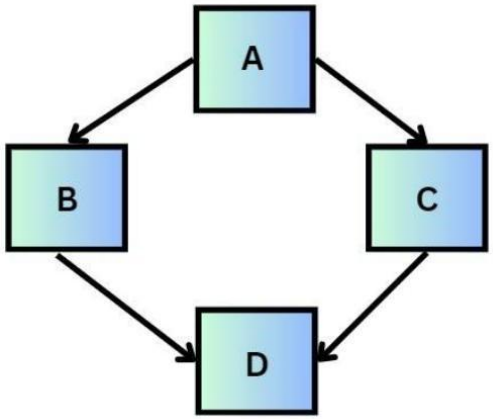
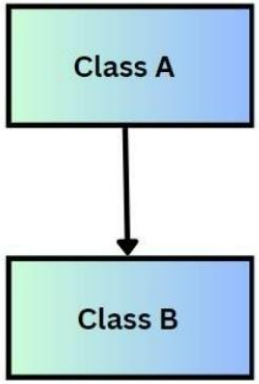


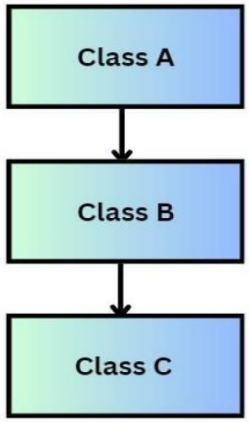
Fig: Inheritance of features in the real world

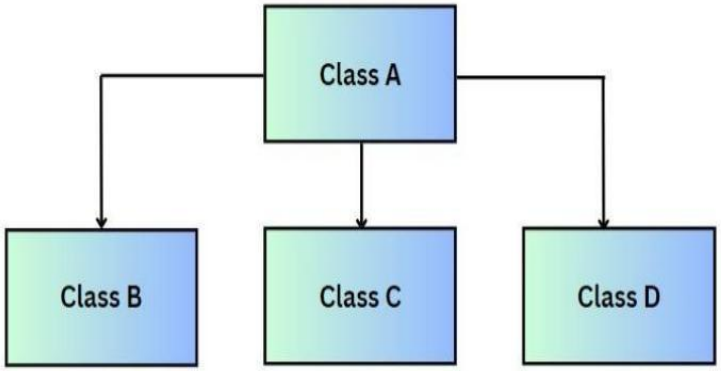
➢ Types of Inheritance:

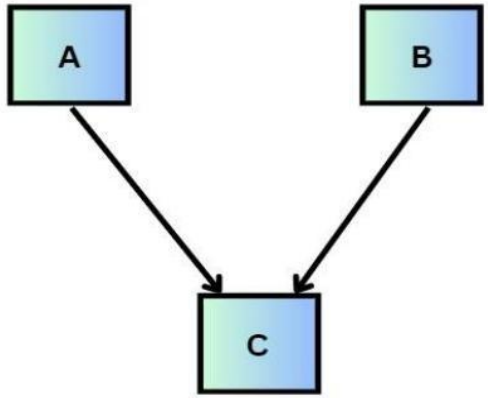
- **Single Inheritance -** A child class inherits from a single parent class.
- **Multilevel Inheritance** - A class inherits from another class, which itself is a subclass of another class.
- **Hierarchical Inheritance** - Multiple child classes inherit from a single parent class.
- **Hybrid Inheritance** – Combination of two or more types of inheritance
- **Multiple Inheritance -**


A
B
C
D


Class A
Class B


Class A
Class B
Class C


Class A
Class B
Class C
Class D


A
B
C

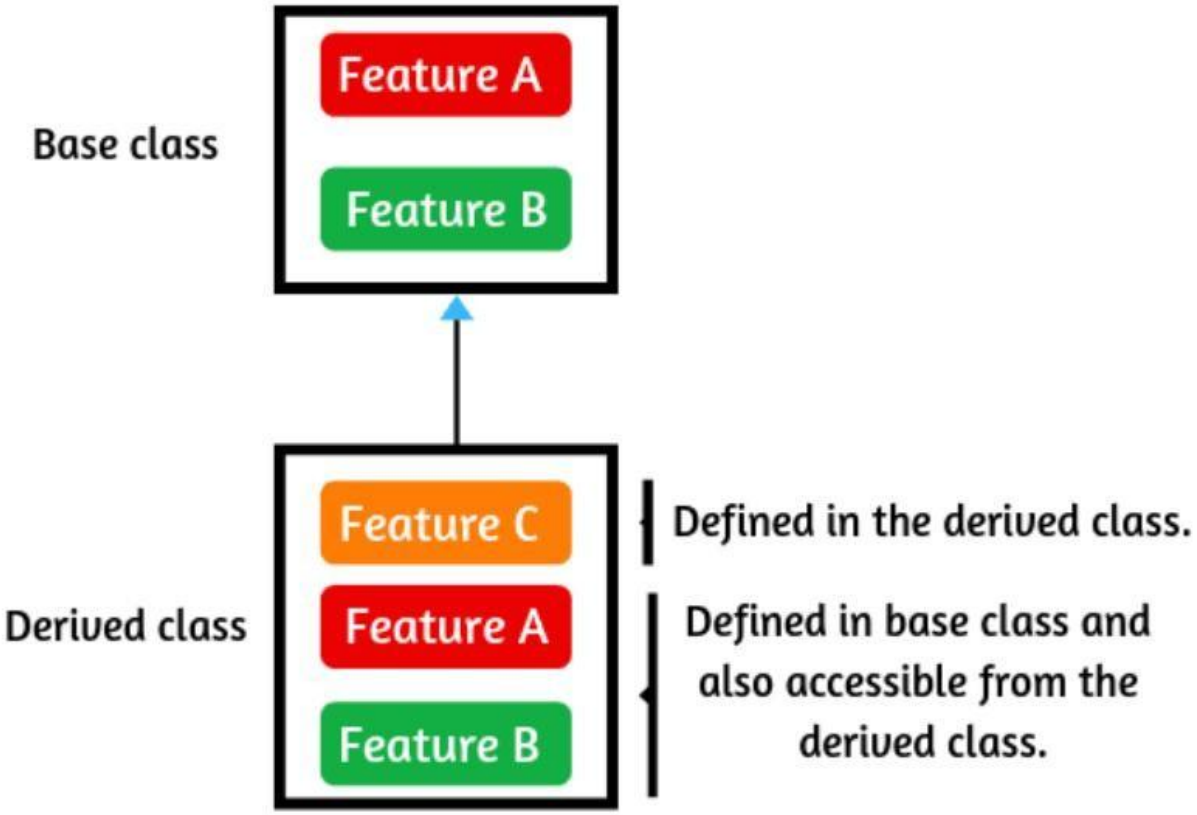


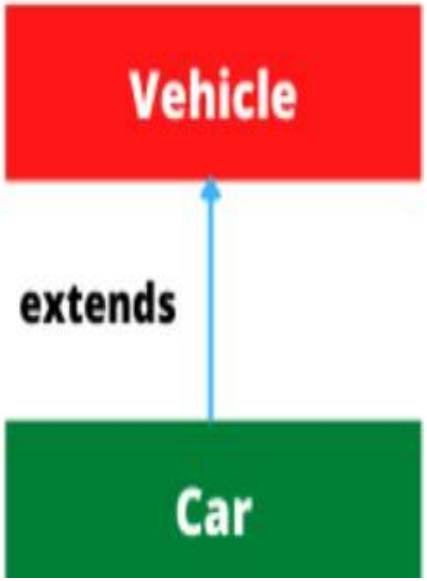
Fig: Base class and Derived class relationships

Syntax

```
public class Vehicle {
    ..............
    ..............
 }
```

IS-A Relationship

```
public class Car extends Vehicle {
    ..............
    ..............
}
```



- **Compile-Time Polymorphism (Method Overloading):**

Methods have the same name but different parameter lists (different signatures).

- **Runtime Polymorphism (Method Overriding):**

A child class provides its implementation of a method defined in the parent class.

Rules of overloading and overriding

- **Overloading:**

1. Method name must be the same, but parameters must differ.
2. Can have different return types but must differ in parameters.
3. Happens in the same class.

- **Overriding:**

1. Method signature (name, parameters) must be the same.
2. Access modifiers cannot reduce visibility.
3. Happens between parent and child classes.

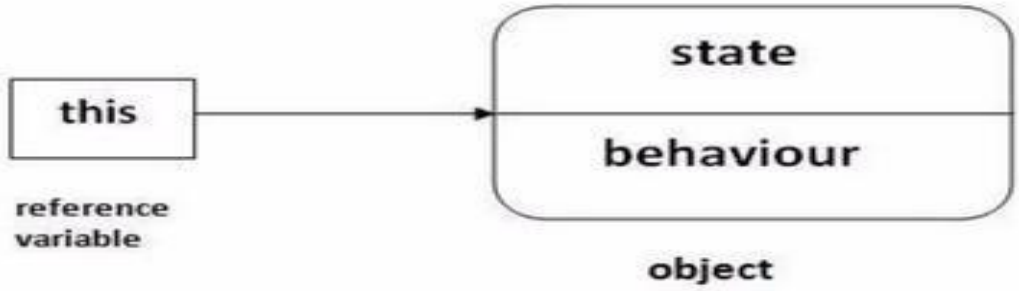
Property	Overloading	Overriding
Method name	Must be same	Must be same
Argument type	Must be different	Must be same(including order)
Method signature	Must be different	Must be same
Return type	No restriction	Can be different
Also known as	Static polymorphism / compile time polymorphism/ Early binding	Dynamic polymorphism/ Run time polymorphism/ Late binding
Inheritance	May or may not be required	Must be required
private , static, final method	Can be overloaded	Can not be overridden

this keyword

- this keyword refers to the current object of a class . It is used with in a method or constructor to explicitly access the current instance variable of the object.
- The primary purpose of this is to resolve ambiguity.

➢ Uses:-

- this keyword can be used to refer **current class instance variable**.
- this keyword can be used to invoke current class method (implicitly).
- The only applicable access modifier for constructor are public, private , protected , default



super keyword

- Super can be used to invoke parent class method [super.methodname()]
- Super used to refer immediate parent class instance variable [super.variablename]
- Super is used to invoke parent class constructor [super()]

super,super() and this,this()

super(),this()	super,this
These are constructor calls to calls super class and current class constructors	These are keywords to refer super class and current class instance members
We can use only in constructors as first line	We can use anywhere except static area
We can use only once in constructor	We can use any number of times.

Constructor Chaining

- calling one constructor from another constructor in the same class or from a parent class.
- Chaining within the same class using this()- Example
- Chaining between parent and child class using super() – Example

NOTE-

- this() → calls another constructor of the same class
- super() → calls parent class constructor
- Must be first line in constructor
- Only one constructor call (this or super) allowed per constructor

Static and non- static concept of method and variable

Association, Aggregation and Composition

- **Inheritance** — an **"is a"** relationship
- **Association** — a **"has a"** relationship
- **Composition** and **Aggregation** are the two forms of association.

Composition is defined by the PART-OF relationship which means that one object IS PART-OF ANOTHER OBJECT, but Aggregation is defined by the HAS-A relationship which means that one object HAS-A RELATION with another object.

HAS-A Relationship (Association)

- Definition:

  One class **contains** or **owns** a reference to another class or data member.

- Example:

  Student **HAS-A** name

  Student **HAS-A** rollno

  Car **HAS-A** Engine

- Achieved by:

  Declaring **object references** as instance variables inside a class.

HAS-A Relationship (Association)

- Type:

1. Aggregation

weak relationship between classes.

Ex – Car has a Music player

Student has a laptop

2. Composition

strong relationship between classes.

Ex - Car has a Engine

Student has a Address

IS-A Relationship (Inheritance)

- Example –

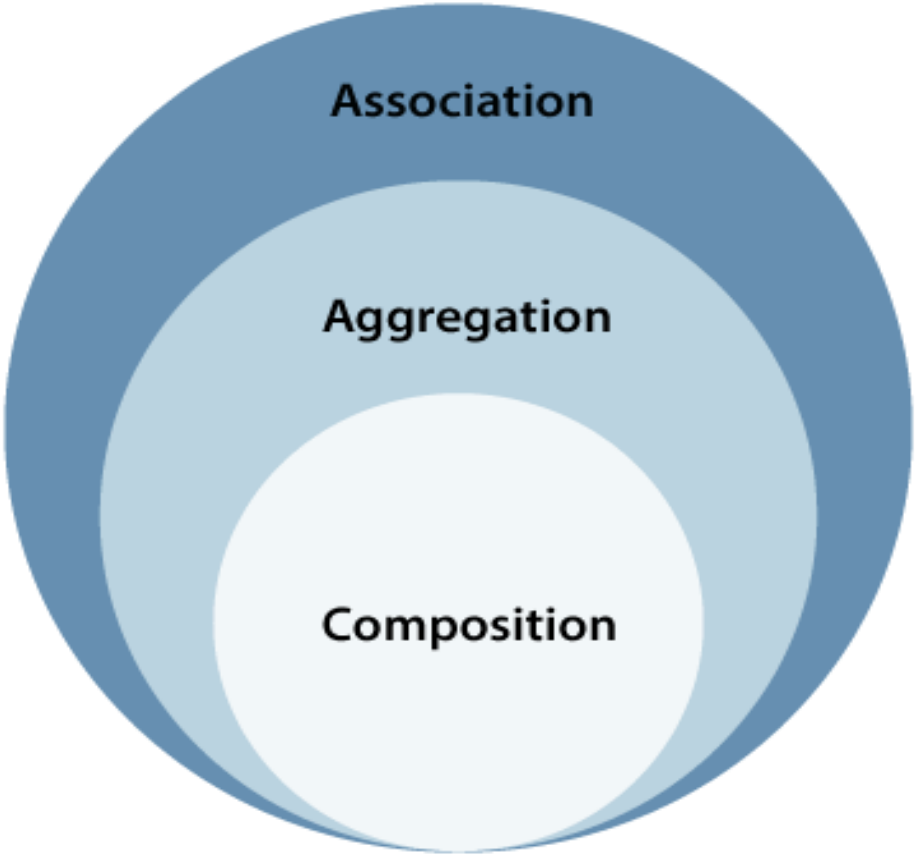
  Car IS-A Vehicle
- Achieved by:

  using extends or implements keyword

NOTE-

Association (has a relationship) – classes loosely coupled

Inheritance (Is a relationship) – classes tightly coupled


Association
Aggregation
Composition

Reference Assignment Compatibility

- A **parent class reference** can refer to a **child class object**.
- A **child class reference CANNOT refer to a parent class object**

1. Same type reference ✓
2. Parent reference → Child object ✓
3. Child reference → Parent object 

Example(a)

```
Parent p1 = new Parent();
Child  c1  = new Child();
```

Example (b)

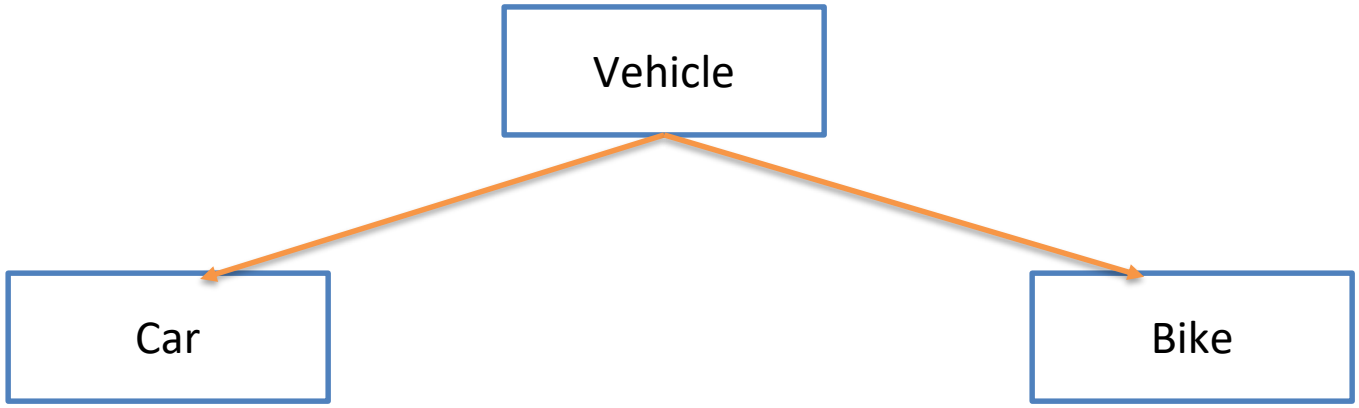
```
Parent p2 = new Child();
```

But only parent member are accessible

Example (c)

```
Child c2 = new Parent ();
```

Example



Vehicle- startEngine()
stopEngine()
Car – openSunroof()
closeSunroof()
Bike - kiskStart()

Example- Vehicle vehicle = new Car();
vehicle.startEngine(); ✓
vehicle.openSunroof(); ✗

Assignments

1. Create a class with a method that prints "parent class" and its subclass with another method that prints " child class". Now, create an object for each of the class and call

   - method of parent class by object of parent class

   - method of child class by object of child class

   - method of parent class by object of child class

2. Create a class to print the area of a **square** and a **rectangle**. The class has two methods with the same name but different number of parameters. The method for printing area of rectangle has two parameters which are length and breadth respectively, while the other method for printing area of square has one parameter which is side of square.

3. Create an class ‘**Hospital**’ provides no. of patients admitted in it. But number of patients varies with the different hospitals.‘**Hospital**’ parent class which has one method getNumberOfPatients() and create sub classes ‘ **Fortis**’,‘**Narayana**’ and ‘**Apolo**’ class which extends parent class & override its method(getNumberOfpatients()). (Method overriding)

4. Create a class named **'Member**' having the following members:

   data members(variables)

   - Name
   - Age
   - Address
   - Salary

   Two classes **'Emp**' and **'Manager**' inherits the **'Member**' class. The 'Employee' and 'Manager' classes have data members 'shift' and 'department' respectively. Now, assign name, age, address and salary to an employee and a manager by making an object of both of these classes and print the same.

5. Create a class named **'Shape**' with a method to print "shape". Then create two other classes named **'Rectangle**', **'Circle**' inheriting the Shape class, both having a method to print " rectangular shape" and "circular shape" respectively. Create a subclass **'Square**' of **'Rectangle**' having a method to print "Square is a rectangle". Now call the method of 'Shape' and 'Rectangle' class by the object of 'Square' class.

6. Create a class **'Degree**' having a method ' getDegree' that prints "I got a degree". It has two subclasses namely **'Undergraduate**' and **'Postgraduate**' each having a method with the same name that prints "Undergraduate" and "Postgraduate" respectively. Call the method by creating an objects of subclasses respectively.

7. A boy has his money deposited Rs10000, Rs15000 and Rs20000 in banks-Bank A, Bank B and Bank C respectively. We have to print the money deposited by him in a particular bank. Create a class **'Bank**' with a method 'getBalance' which returns 0. Make its three subclasses named **'BankA**', **'BankB**' and **'BankC**' with a method with the same name 'getBalance' which returns the amount deposited in that particular bank. Call the method 'getBalance' by the object of each of the three banks.

8. A school allows student registration with different levels of information:

- Only name
- Name and roll number
- Name, roll number, and grade

Write a program using constructor overloading to handle all cases.

9. An electricity board calculates bills in different ways:-

- Based on units consumed
- Units + fixed charge
- Units + fixed charge + tax

Write a program using method overloading to calculate the bill.

10. A transport system stores vehicle information. All vehicles have a method **getFuelType().**Different vehicles use different fuels.
Write a program where:

- Vehicle is the base class
- Car and Bike override the fuel type method

11. A bank account is created with an account number and balance.
Money can be deposited:

- Using cash
- Using cheque

Write a program using:

- Constructor to initialize account
- Method overloading for deposit

12. A school system stores people information . Each person has a role.
Students and teachers have different roles.
Write a program using:

- Constructor to initialize person data
- Method overriding for role

THANK YOU!!