SNS COLLEGE OF ENGINEERING
Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NAAC – UGC with 'A' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

23CSB101
OBJECT ORIENTED PROGRAMMING

UNIT II
INHERITANCE, PACKAGES AND INTERFACES

INHERITANCE

By
M.Kanchana
Assistant Professor/CSE

INHERITANCE

Inheritance is a process of deriving a new class from existing class, also called as “**extending a class**”.
When an existing class is extended, the new (inherited) class has all the properties and methods of the existing class and also possesses **its own** characteristics.

The class whose property is being inherited by another class is called **“base class” (or) “parent class” (or) “super class”.**

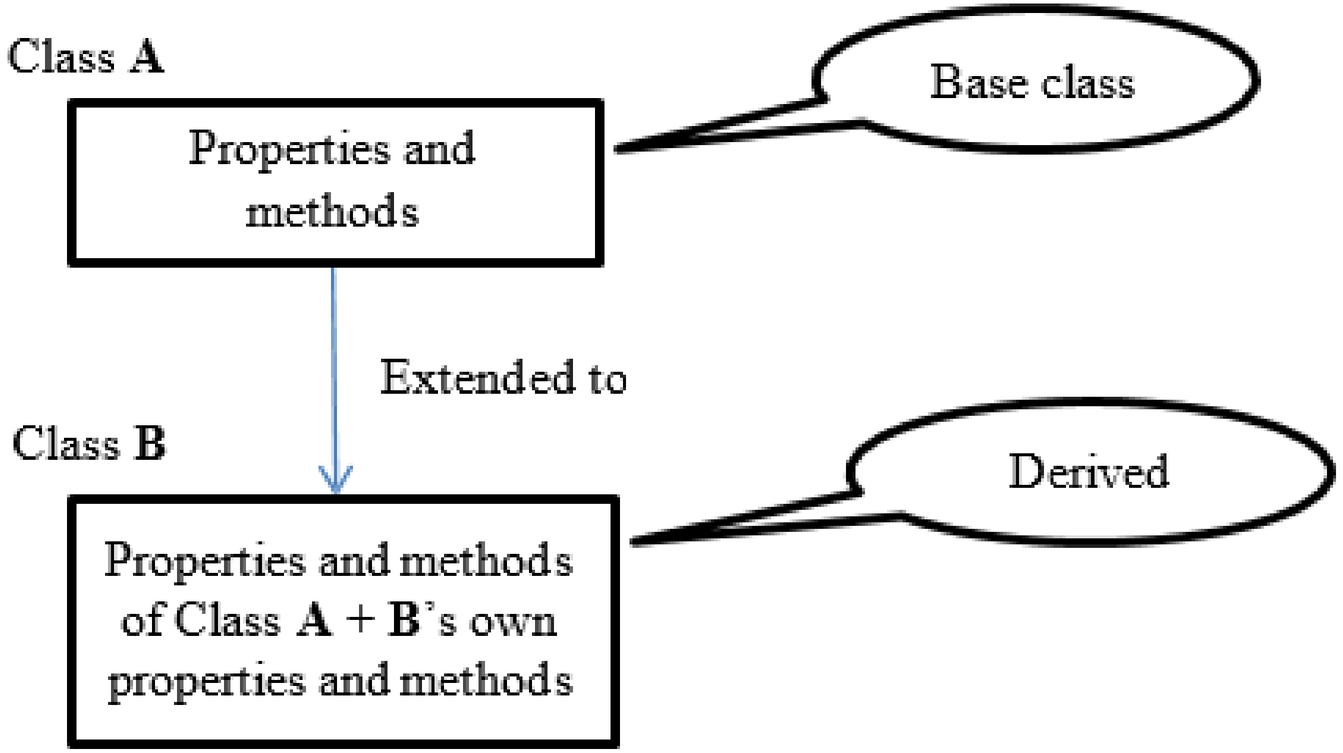
The class that inherits a particular property or a set of properties from the base class is called **“derived class” (or) “child class” (or) “sub class”.**

is-a relationship

In Java, inheritance is an is-a relationship. That is, we use inheritance only if there exists an is-a relationship between two classes. For example,

- Car is a Vehicle
- Orange is a Fruit
- Surgeon is a Doctor
- Dog is an Animal

INHERITANCE


Class A
Properties and methods
Base class
Extended to
Class B
Derived
Properties and methods of Class A + B's own properties and methods

INHERITANCE

```
[access_specifier] class subclass_name extends superclass_name
{
        // body of class
}
```

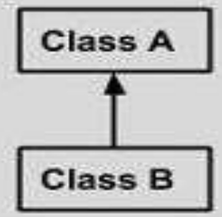
INHERITANCE

Characteristics of Class Inheritance:

1. A class cannot be inherited from more than one base class. Java does not support
the inheritance of multiple super classes into a single subclass.
2. Sub class can access only the non-private members of the super class.
3. Private data members of a super class are local only to that class. Therefore, they can’t be accessed outside the super class, even sub classes can’t access the private members.
4. Protected features in Java are visible to all subclasses as well as all other classes in the same package.

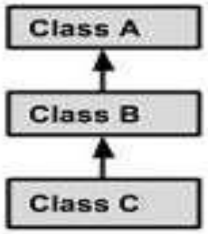
INHERITANCE

Single Inheritance



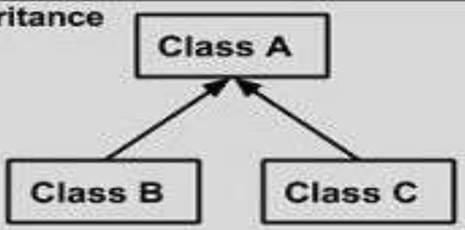
```
public class A {
   ........
}
public class B extends A {
   .........
}
```

Multi Level Inheritance



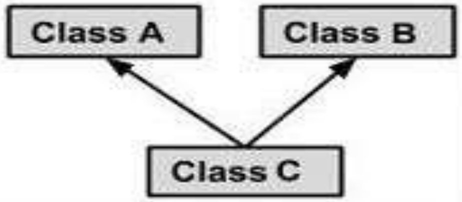
```
public class A { ....................}
public class B extends A {....................}
public class C extends  B {..................... }
```

Hierarchical Inheritance



```
public class A { ....................}
public class B extends A {....................}
public class C extends A {..................... }
```

Multiple Inheritance



```
public class A { ....................}
public class B {....................}
public class C extends  A,B {
  .......................
} // Java does not support mutiple Inheritance
```

INHERITANCE

Why multiple inheritance is not supported in java?

- Multiple inheritance means a class inherits from more than one parent class.
- If two parent classes have the same method, and a child class inherits from both, which method should it use?

```
   CEO (Parent Class)
        /        \
      M1    M2  ((Managers M1 & M2) Both have "giveTask()" method)
        \        /
         Employee  (Child Class - Confused which giveTask() to execute!)
```

INHERITANCE

```
class LandAnimal {
    void move() {
        System.out.println("Moves by walking."); }}

class WaterAnimal {
    void move() {
        System.out.println("Moves by swimming.");}}

class Frog extends LandAnimal, WaterAnimal {

    // ? Which move() method should be inherited?
}
public class Main {
    public static void main(String[] args) {
        Frog frog = new Frog();
        frog.move();  // ✕ Java would not know which move() to call!
    }
}
```

INHERITANCE

What is an **Interface** in Java?

- An interface in Java is a blueprint for a class that only contains abstract methods (by default) and constants.
- It is used to achieve 100% abstraction and **multiple inheritance** in Java

```
interface Animal {
    void makeSound(); // Abstract method (no body)
}
```

.

An **interface** allows us to switch between different database implementations **without modifying the main application code**.

This is possible because the application depends on an **interface** rather than a specific database class.

INHERITANCE

```
interface LandAnimal {
    void move(); // Walk on land
}
interface WaterAnimal {
    void move(); // Swim in water
}
class Frog implements LandAnimal, WaterAnimal {
        public void move() {
        System.out.println("Frog jumps on land and swims in water.");}}

public class Main {
    public static void main(String[] args) {
        Frog frog = new Frog();
        frog.move();  //  Output: Frog jumps on land and swims in water.
    }
}
```

THANK YOU