# INHERITANCE IN JAVA

-Compiled by Nikahat Mulla

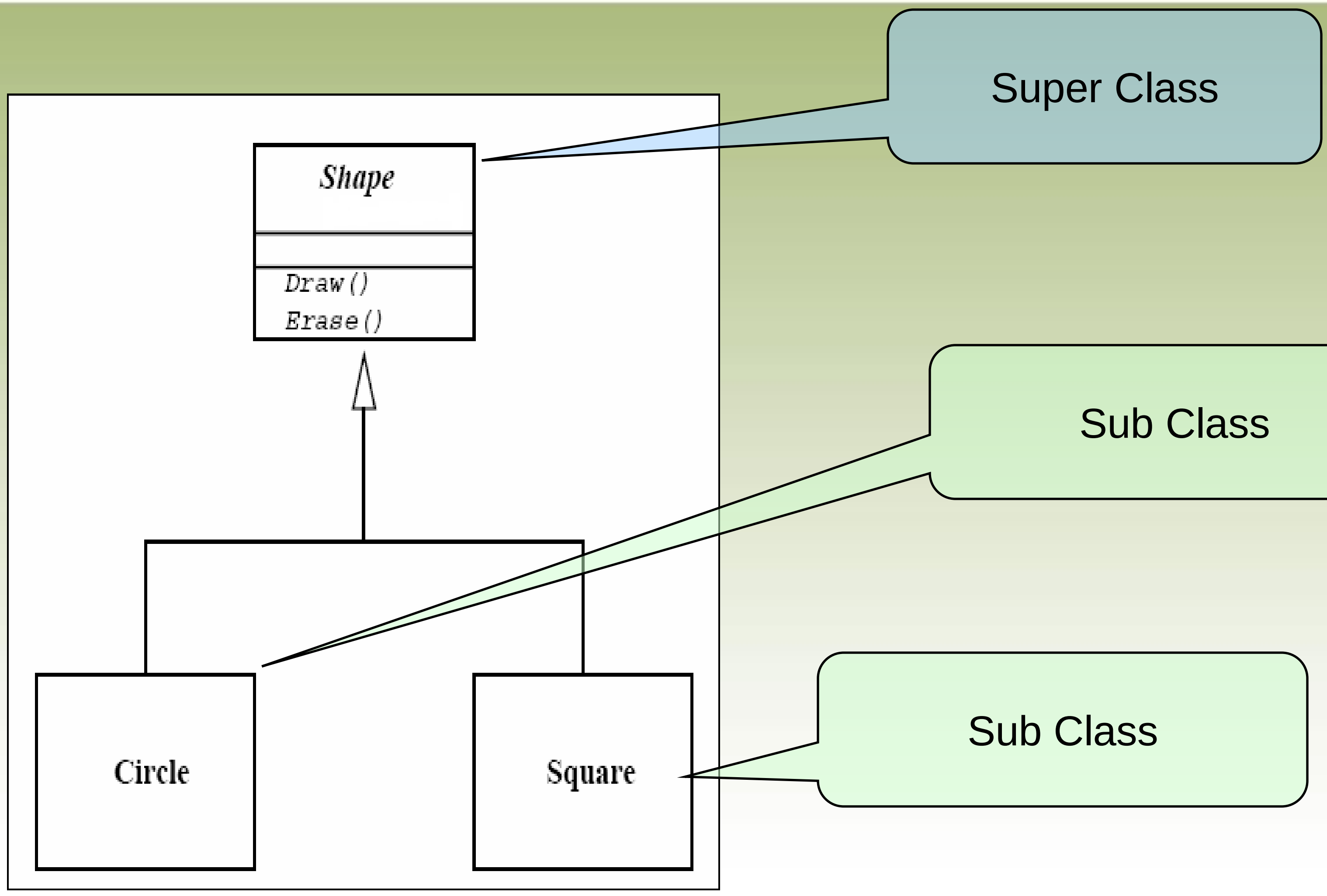
## Agenda

- Inheritance
  - Introduction
  - Types of Inheritance
  - Polymorphism
    - Method Overloading and Overriding

# Inheritance

- Capability of a class to use the properties & methods of another class while adding its own functionality
- A class derived from another class is called as ***subclass / derived class / extended class / child class***
- The class from which the subclass is derived is called as ***superclass / base class / parent class***
- Each class is ***allowed to have one direct superclass***
- Each ***superclass*** can have ***unlimited number of subclasses***

# Inheritance (Contd…)

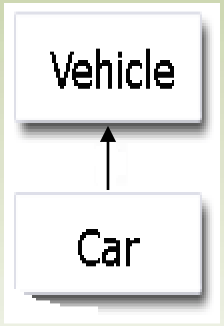


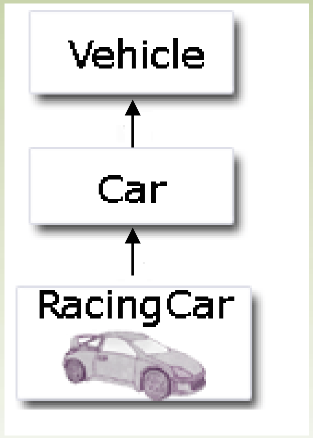
## Types of Inheritance

- Java supports two types of inheritance

Single Inheritance

Multilevel Inheritance





Note: Java doesn't support Multiple Inheritance (One sub class deriving from more than one super classes) through classes but through interfaces

## Example 1

- Consider a Mobile Phone Model: Example: Redmi 9
- If I want to add more features to Redmi 9 to develop my new model, Redmi 9 Pro, I reuse the basic design of Redmi 9 and add the additional features, rather than creating Redmi 9 Pro from scratch.
- This concept is called Inheritance

## Example 2: Constructing a Table

- Consider a class called TableTop. It has features: color, dimensions of table.
- Let’s say you want to now create Table objects.
- One way to do this is, since, we already have TableTop class we can reuse it using inheritance, add features to it to make the complete Table.
- Let’s Design it.

# Class TableTop ← Class Table

```
class TableTop{
        private float length,breadth;
        private String color;
        TableTop(){
            length=breadth=1;
             color="Black";
        }
         TableTop(float l,float b,String
c){
             color=c;
             length=l;
             breadth=b;
         }
         public float area(){
                  return length*breadth;
         }
         public String getColor(){
                  return color;
         }
}
```

```
class Table extends TableTop{
        int no_legs;
        float height;
        Table(){
                  no_legs=4;
                  height=2.5f;
         }
         Table(int nl,float h,float l,float b,String c){
                  super(l,b,c);
                  no_legs=nl;
                  height=h;
         }
         public void print(){
               System.out.printf("Area=%.2f\n",area());
               System.out.println("No. of legs="+no_legs);
               System.out.printf("Height=%.2f\n",height);
               System.out.println("Color="+getColor());
         }
}
```

## Main Class for testing

```
class TestTable{
        public static void main(String []args){
                Table t1=new Table(3,4,5,6,"Gray");
                t1.print();
                Table t2=new Table();
                t2.print();
        }
}
```

## Constructors in Inheritance

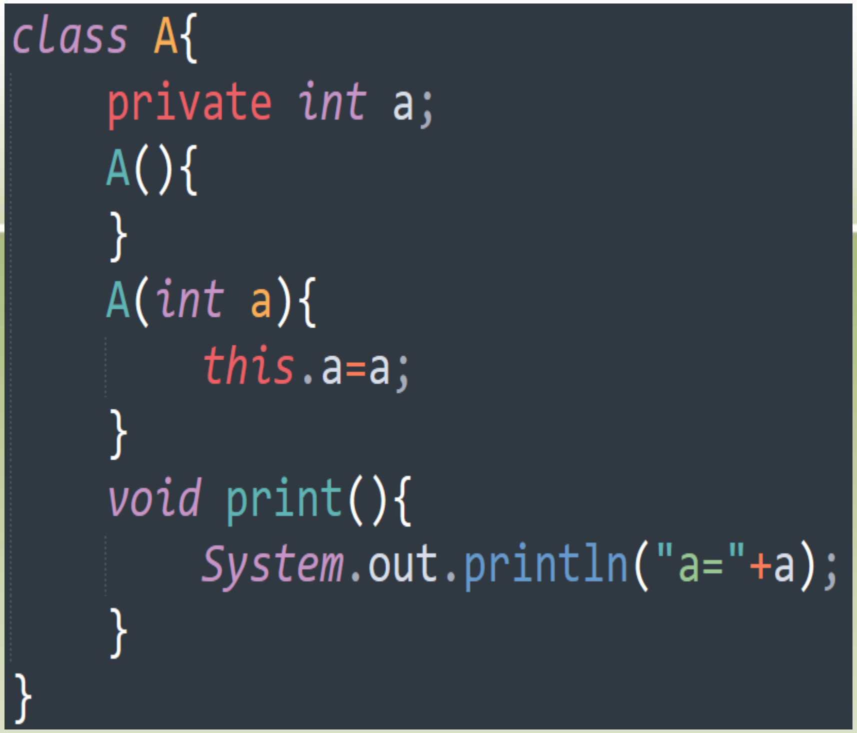
- Constructors are invoked in the order of hierarchy
- While instantiating a sub class, its super class default constructor will be invoked first, followed by the sub class constructor
- The keyword super can be used to invoke the super class parameterized constructor instead of the default
- Remember that:
  - super() call must occur as the first statement in constructor
  - super() call can only be used in a constructor definition

## Using *super*

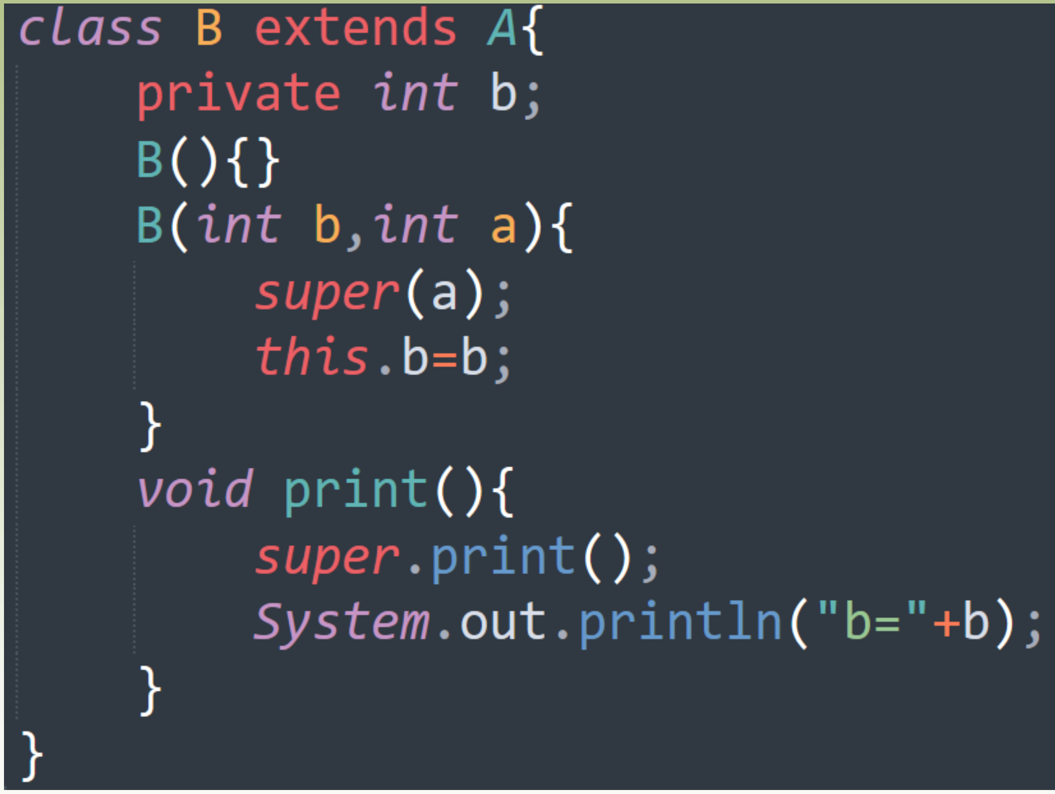
- The keyword super refers to members of the super class

- *U*sed when member names of the subclass hide members by the same name in the super class

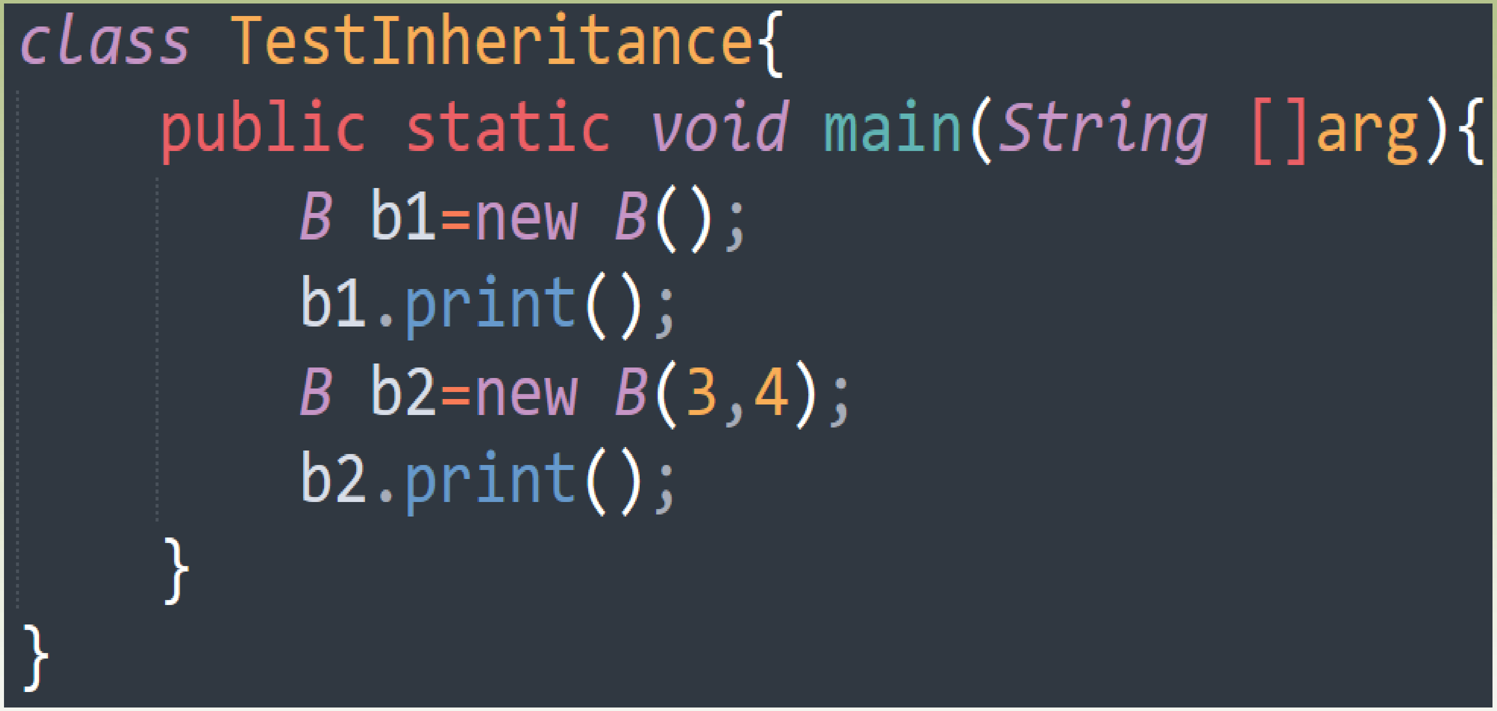
- super.member
  - *member* can be either a method or an instance variable



```java
class A{
    private int a;
    A(){
    }
    A(int a){
        this.a=a;
    }
    void print(){
        System.out.println("a="+a);
    }
}
```



```java
class B extends A{
    private int b;
    B(){}
    B(int b,int a){
        super(a);
        this.b=b;
    }
    void print(){
        super.print();
        System.out.println("b="+b);
    }
}
```



```
class TestInheritance{
    public static void main(String []arg){
        B b1=new B();
        b1.print();
        B b2=new B(3,4);
        b2.print();
    }
}
```

## Polymorphism

- Means one entity having many (poly) forms (morph)

- We can have multiple methods with the same name in the same class, i.e. Method Overloading

- Capability of an action or *method* to do different things based on the object that it is acting upon, i.e. Method Overriding

## Method Overloading

- Two methods in a class can have same name, provided their signature is different

```
class Test {
    public static void main(String args[]) {
        myPrint(5);
        myPrint(5.0);
    }

    static void myPrint(int i) {
        System.out.println("int i = " + i);
    }

    static void myPrint(double d) {

       System.out.println("double d = " + d);
    }
}
```

Same name with different parameters

## Method Overriding (Contd…)

- Useful if a derived class needs to have a different implementation of a certain method from that of the superclass
- A subclass can override a method defined in its superclass by providing a new implementation for that method
- The new method definition must have the same method signature (i.e., method name & parameters) and return type
- The new method definition cannot narrow the accessibility of the method, but it can widen it

## Using Parent class reference to call child class methods

- We can assign a variable of a certain class type with an instance of that particular class or an instance of any subclass of that class

```
public class Mammal { .. . . .  }
public class Dog extends Mammal {  . . . .  }
public class Cat extends Mammal {  . . . .  }
Mammal m;//parent class reference
m = new Dog( );//m contains child class
object
m.speak(); // Invokes Dog’s implementation,
//dogs bark (speaking for a dog is barking-
//same name, different behavior)
m= new Cat();
m.speak(); // Invokes Cat’s implementation,
//cat meows (speaking for a cat is meowing-
//same name, different behavior)
```

## Thank You