



# CSN-103: Fundamentals of Object Oriented Programming

**Instructor: Dr. Rahul Thakur**

Assistant Professor, Computer Science and Engineering, IIT Roorkee



# Superclass Variable, Reference, and Subclass Object



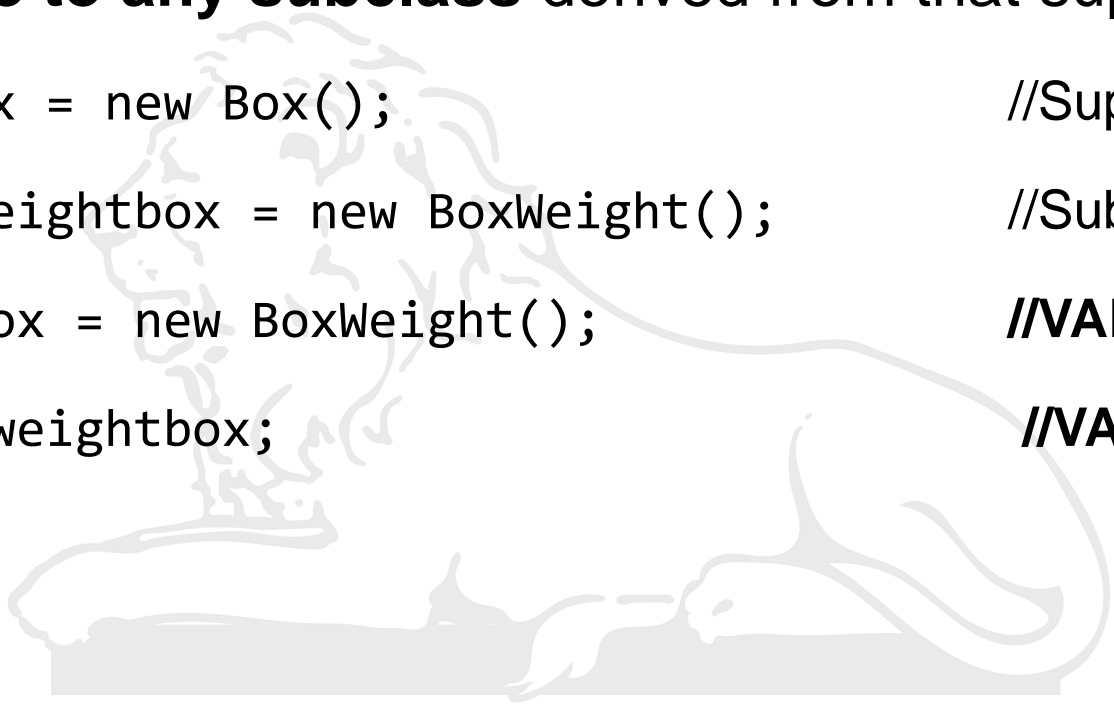
- A reference variable of a **superclass** can be assigned a **reference to any subclass** derived from that superclass

```
Box plainbox = new Box(); //Superclass
```

```
BoxWeight weightbox = new BoxWeight(); //Subclass
```

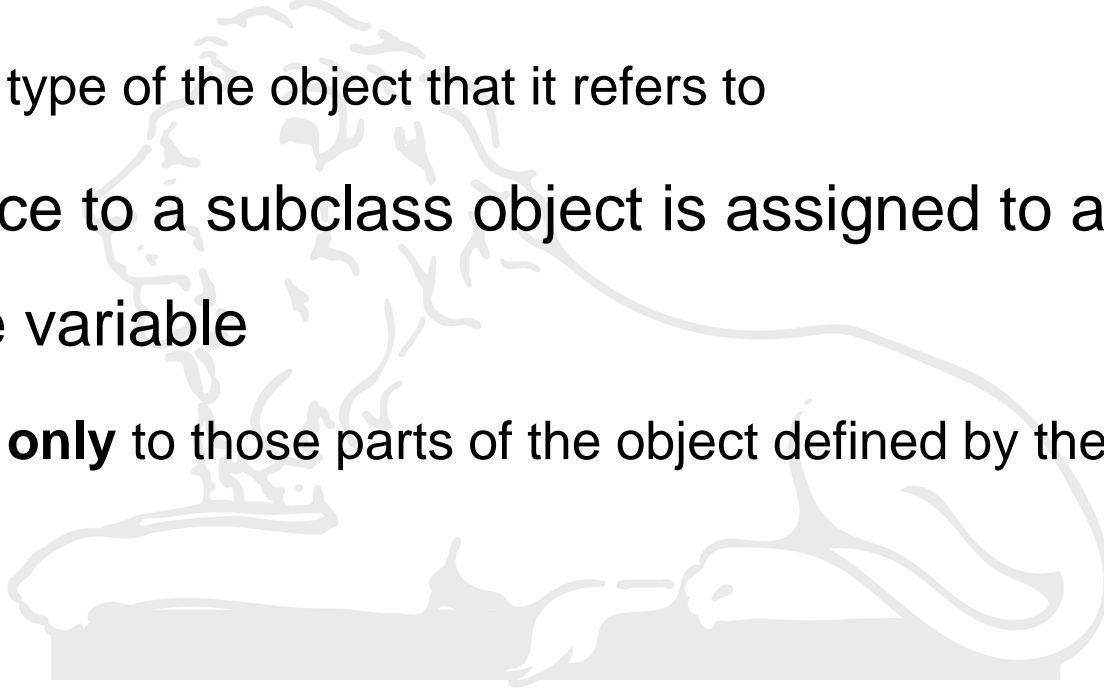
```
Box weightbox = new BoxWeight(); //INVALID
```

```
plainbox = weightbox; //INVALID
```



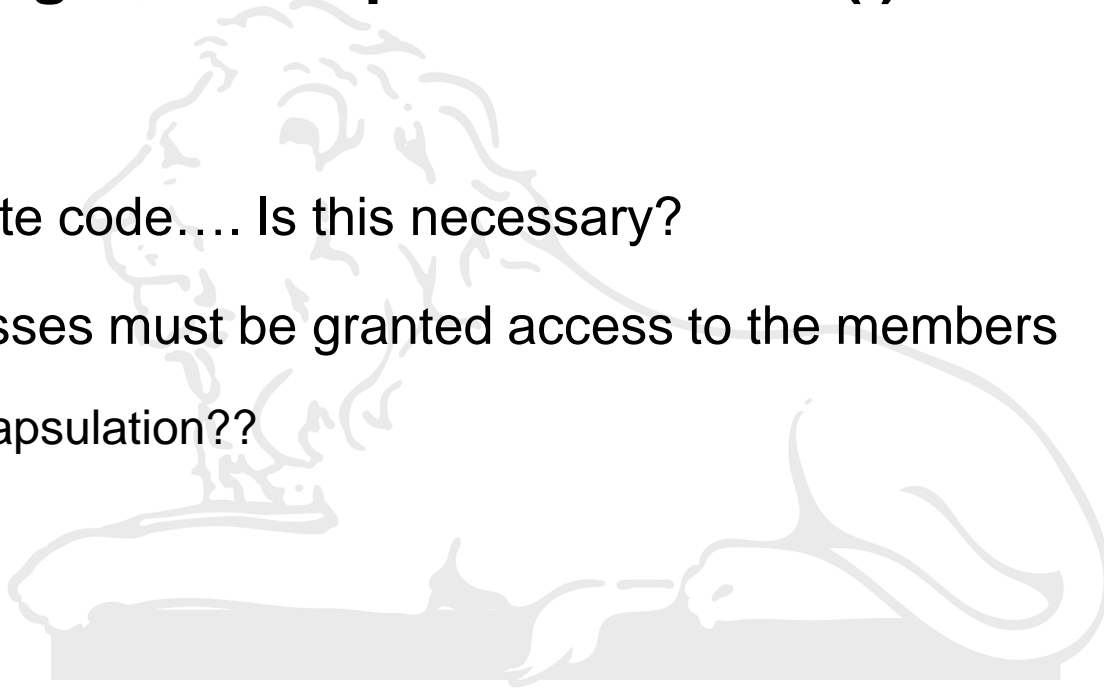
# Referencing a Subclass Object

- The **type of the reference variable** that determines what members can be accessed
  - Not the type of the object that it refers to
- A reference to a subclass object is assigned to a superclass reference variable
  - Access **only** to those parts of the object defined by the superclass



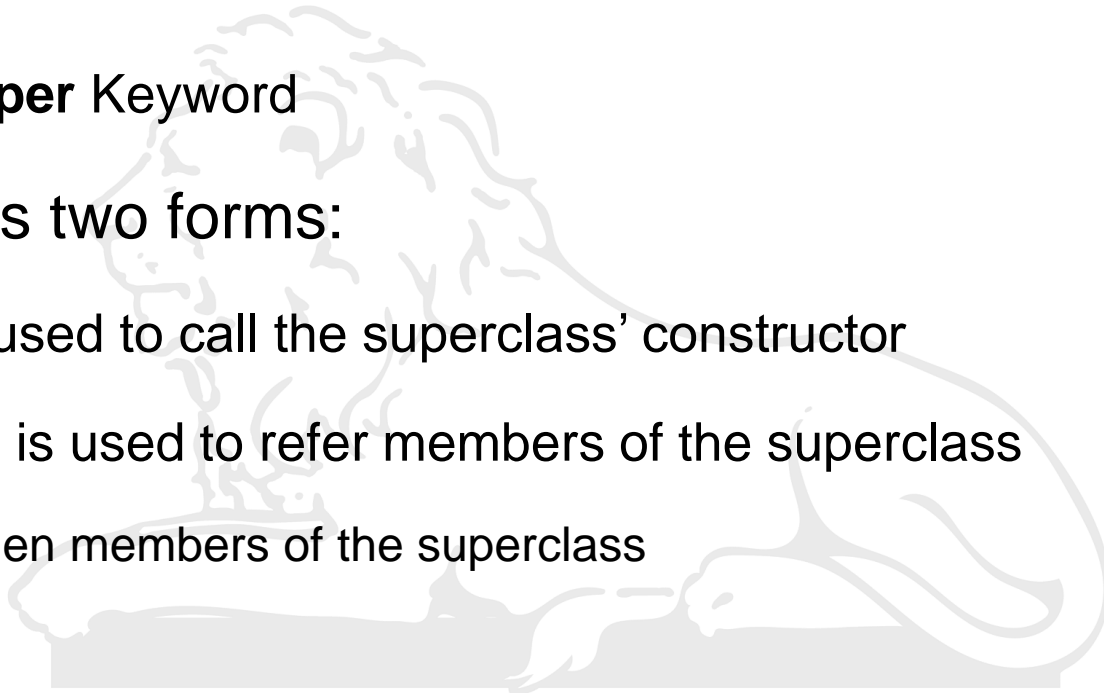
# Keyword super

- The constructor for **BoxWeight** explicitly initializes the **width**, **height**, and **depth** fields of **Box( )**
- Problem:
  - Duplicate code.... Is this necessary?
  - Subclasses must be granted access to the members
    - Encapsulation??



# Keyword super

- Whenever a subclass needs to refer to its **immediate** superclass
  - Use **super** Keyword
- **super** has two forms:
  - One is used to call the superclass' constructor
  - Second is used to refer members of the superclass
    - Hidden members of the superclass



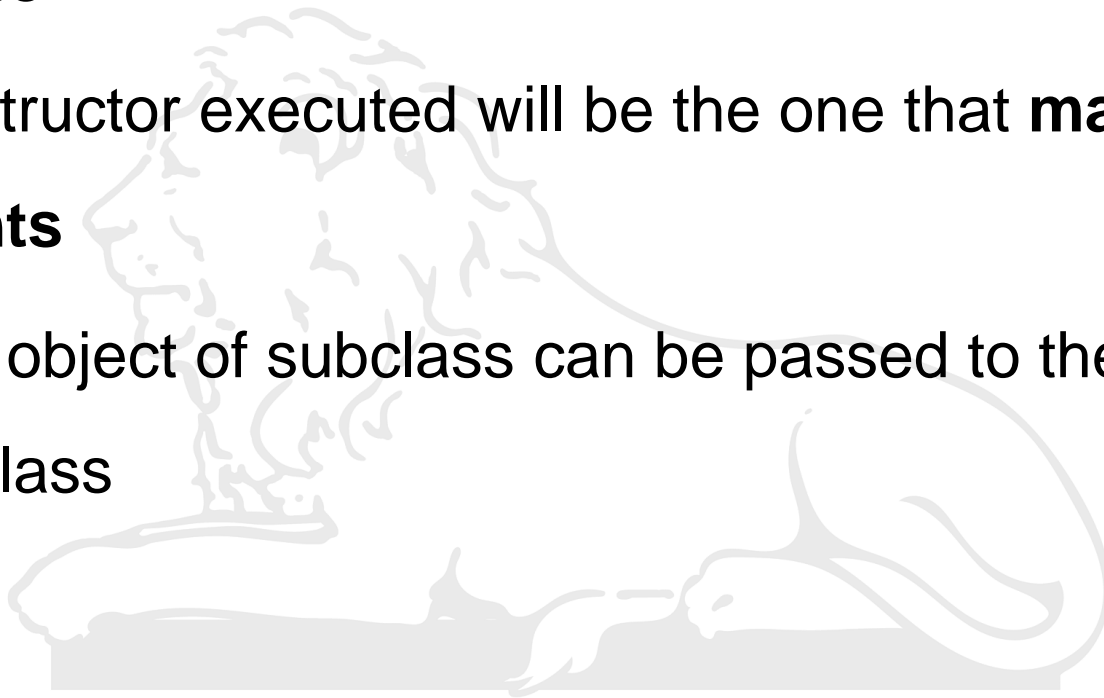
# Superclass Constructor

- A subclass can call superclass constructor  
`super(parameter-list);`
- *super()* must be the **first** statement executed inside subclass' constructor

```
class BoxWeight extends Box {  
    double weight;  
    BoxWeight(double w, double h, double d, double m) {  
        super(w, h, d);  
        weight = m;  
    }  
}
```

# Superclass Constructor

- **super( )** can be called using any form defined by the superclass
- The constructor executed will be the one that **matches the arguments**
- Even the object of subclass can be passed to the constructor of superclass

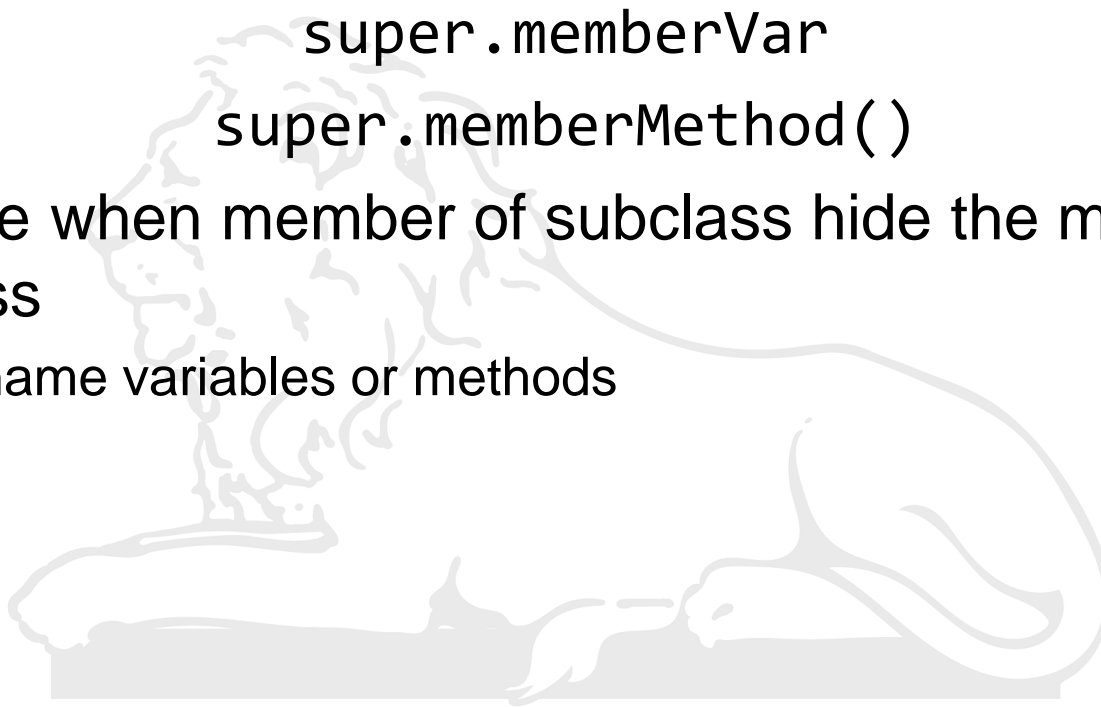


# Superclass Member Access

- Used to access members of superclass
- General form

`super.memberVar`  
`super.memberMethod()`

- Applicable when member of subclass hide the members of superclass
  - Same name variables or methods





# Example

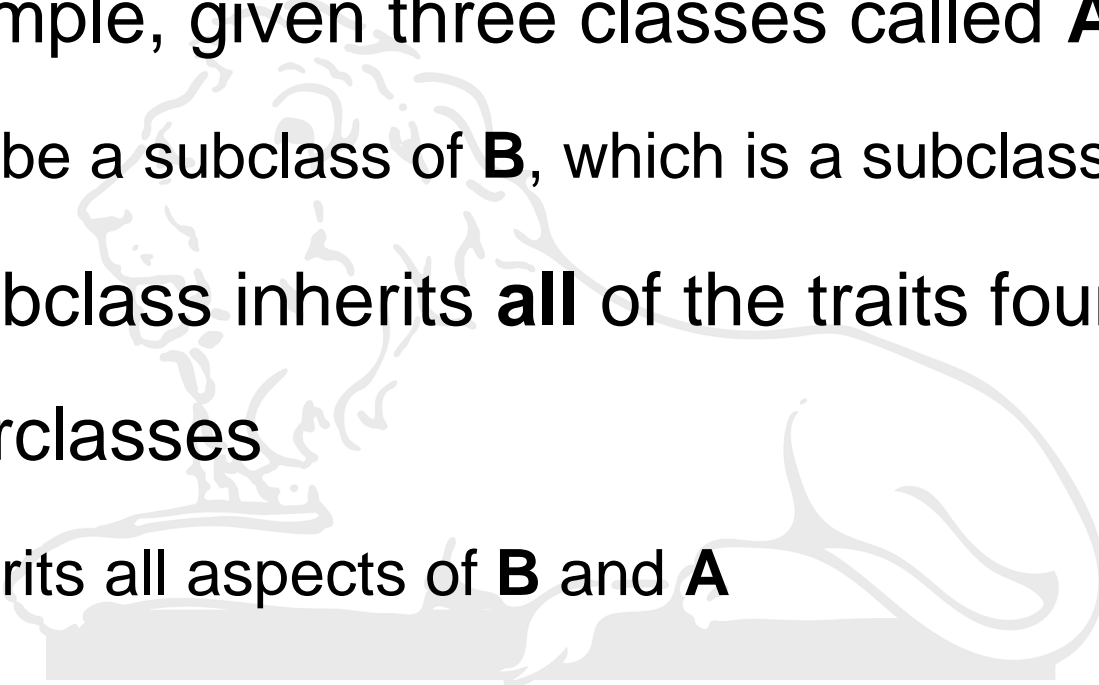


```
class A {
    int i;
}
class B extends A {
    int i;           // this i hides the i in A
    B(int a, int b) {
        super.i = a; // i in A
        i = b;       // i in B
    }
    void show() {
        System.out.println("i in superclass: " + super.i);
        System.out.println("i in subclass: " + i);
    }
}
```

```
class UseSuper {
    public static void main(String args[]) {
        B subOb = new B(1, 2);
        subOb.show();
    }
}
```

# Creating a Multilevel Hierarchy

- Hierarchies: Contain many layers of inheritance
- For example, given three classes called **A**, **B**, and **C**
  - **C** can be a subclass of **B**, which is a subclass of **A**
- Each subclass inherits **all** of the traits found in all of its superclasses
  - **C** inherits all aspects of **B** and **A**



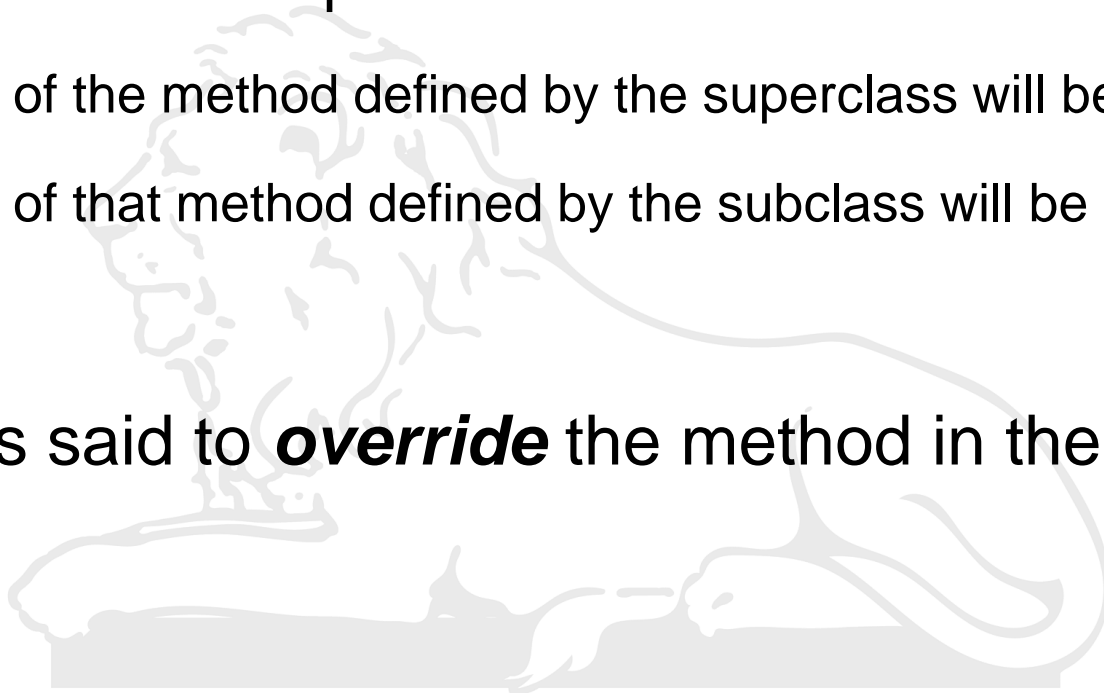
# Constructors and Inheritance

- Class hierarchy: In what order are the constructors for the classes called?
- Constructors are called in order of derivation, from superclass to subclass
- **super( )** must be the first statement executed in a subclass' constructor
- If **super( )** is not used
  - Default or parameter-less constructor of each superclass will be executed

# Method Overriding

- Method in a subclass has the same name and type signature as a method in its superclass
  - Version of the method defined by the superclass will be hidden
  - Version of that method defined by the subclass will be called

Subclass is said to ***override*** the method in the superclass



# super and Method Overriding

- To access the superclass version of an overridden method
  - **super** Keyword: For immediate superclass
- Overloading vs. Overriding
  - Overriding occurs *only* when the names and the type signatures of the two methods are identical
  - Otherwise, the two methods are simply overloaded

