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();             //VALID
plainbox = weightbox;                        //VALID
```

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

```
>javac RefDemo.java
RefDemo.java:68: error: cannot find symbol
                System.out.println("Weight of plainbox is " + plainbox.weight);
                                                                       ^
  symbol:   variable weight
  location: variable plainbox of type Box
1 error
```

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??
    - Access restriction??

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

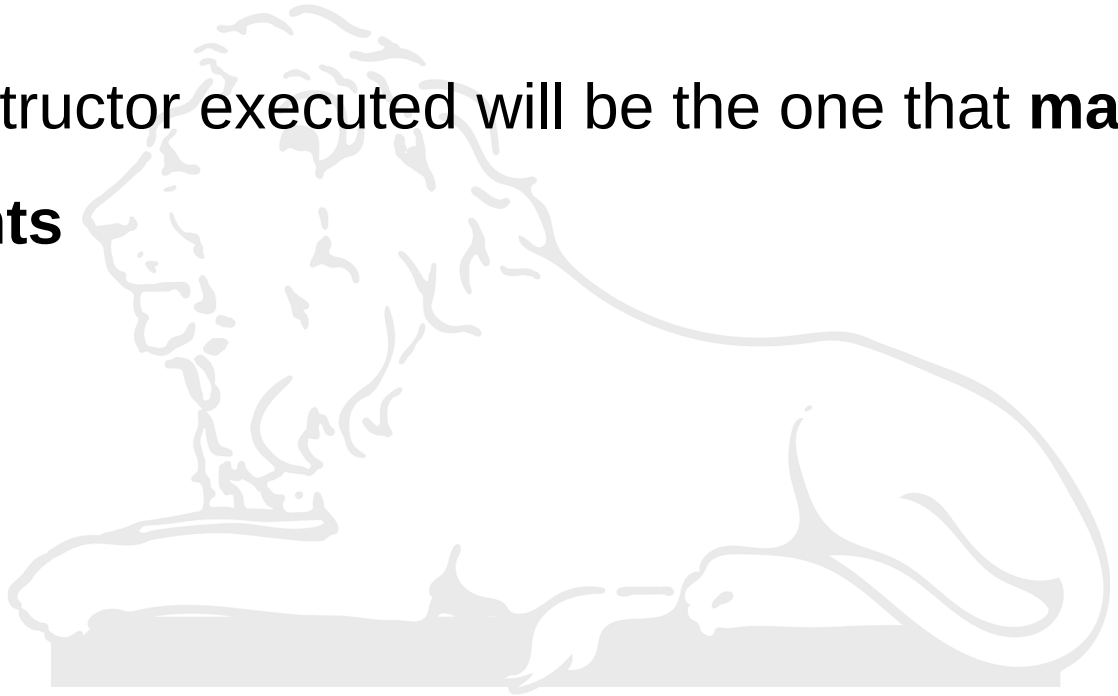
```
super(parameter-list);
```

- *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**



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();
    }
}
```

UseSuper.java

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**

```
G:\My Drive\1.Courses\CSN-103 Object Oriented Programming\Lectures\L22 Programs>java DemoShipment
Box constructor
BoxWeight constructor
Shipment constructor
Volume of shipment1 is 3000.0
Weight of shipment1 is 10.0
Shipping cost: $3.41
```

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 constructor of each superclass will be executed

```
G:\My Drive\1.Courses\CSN-103 Object Oriented Programming\Lectures\L22 Programs>java CallingCons
Inside A's constructor.
Inside B's constructor.
Inside C's constructor.
```