Object Oriented Programming

Topics to be covered today

Inheritance

Inheritance

- One of the pillars of object-orientation.
- A new class is derived from an existing class:
  - existing class is called super-class
  - derived class is called sub-class
- A sub-class is a specialized version of its super-class:
  - has all non-private members of its super-class
  - may provide its own implementation of super-class methods
- Objects of a sub-class are a special kind of objects of a super-class.

Inheritance Syntax

Syntax:

```
class sub-class extends super-class {
…
}
```

Each class has at most one super-class; no multi-inheritance in Java.

No class is a sub-class of itself.

Example: Super-Class & Sub-Class

```
class A {
   int i;

   void showi() {
      System.out.println("i: " + i);
   }

}

class B extends A {

   int j;

   void showj() {
      System.out.println("j: " + j);
   }

   void sum() {
      System.out.println("i+j: " + (i+j));
   }

}
```

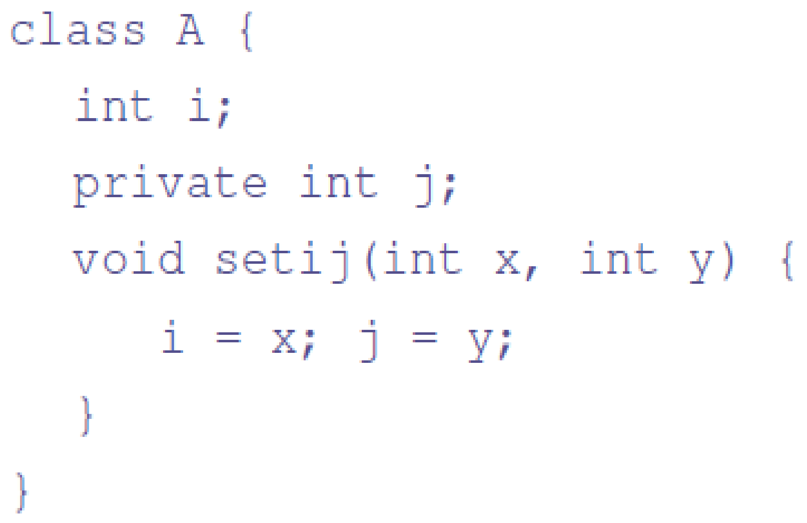
Example: Testing Class

```
class SimpleInheritance {
  public static void main(String args[]) {
     A a = new A();
     B b = new B();
     a.i = 10;
     System.out.println("Contents of a: ");
     a.showi();
     b.i = 7; b.j = 8;
     System.out.println("Contents of b: ");
        b.showi();      b.showj();
     System.out.println("Sum of I and j in b:");
     b.sum();
  }
}
```

Inheritance and Private Members

A class may declare some of its members private.

A sub-class has no access to the private members of its super-class:



```
class A {
   int i;
   private int j;
   void setij(int x, int y) {
      i = x; j = y;
   }
}
```

Inheritance and Private Members

Class B has no access to the A’s private variable j.

This program will not compile:

```
class B extends A {
   int total;
   void sum() {
      total = i + j;
   }
}
```

Example: Box class

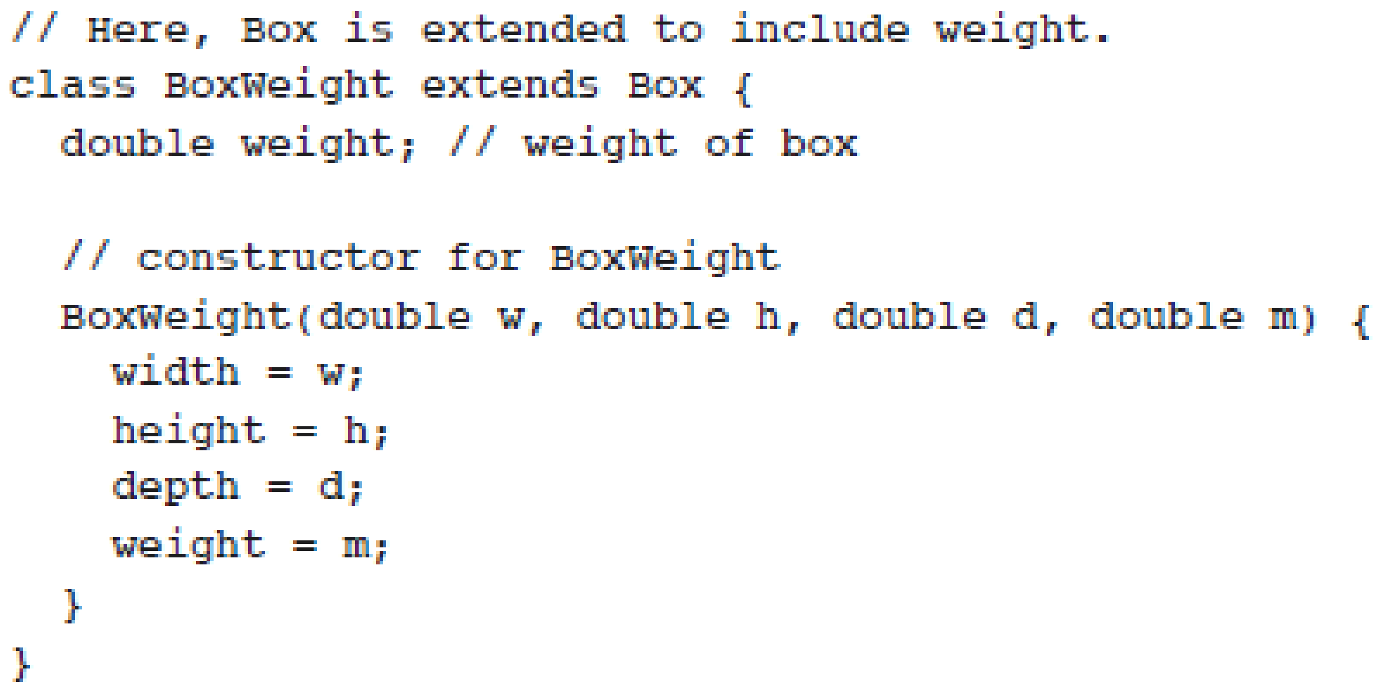
The basic Box class with width, height and depth:

```
// This program uses inheritance to extend Box.
class Box {
  double width;
  double height;
  double depth;

  // construct clone of an object
  Box(Box ob) { // pass object to constructor
    width = ob.width;
    height = ob.height;
  depth = ob.depth;
}

// constructor used when all dimensions specified
Box(double w, double h, double d) {
  width = w;
  height = h;
  depth = d;
}
```

```
// constructor used when no dimensions specified
Box() {
  width = -1;  // use -1 to indicate
  height = -1; // an uninitialized
  depth = -1;  // box
}

// constructor used when cube is created
Box(double len) {
  width = height = depth = len;
}

// compute and return volume
double volume() {
  return width * height * depth;
}
```

Example: BoxWeight class

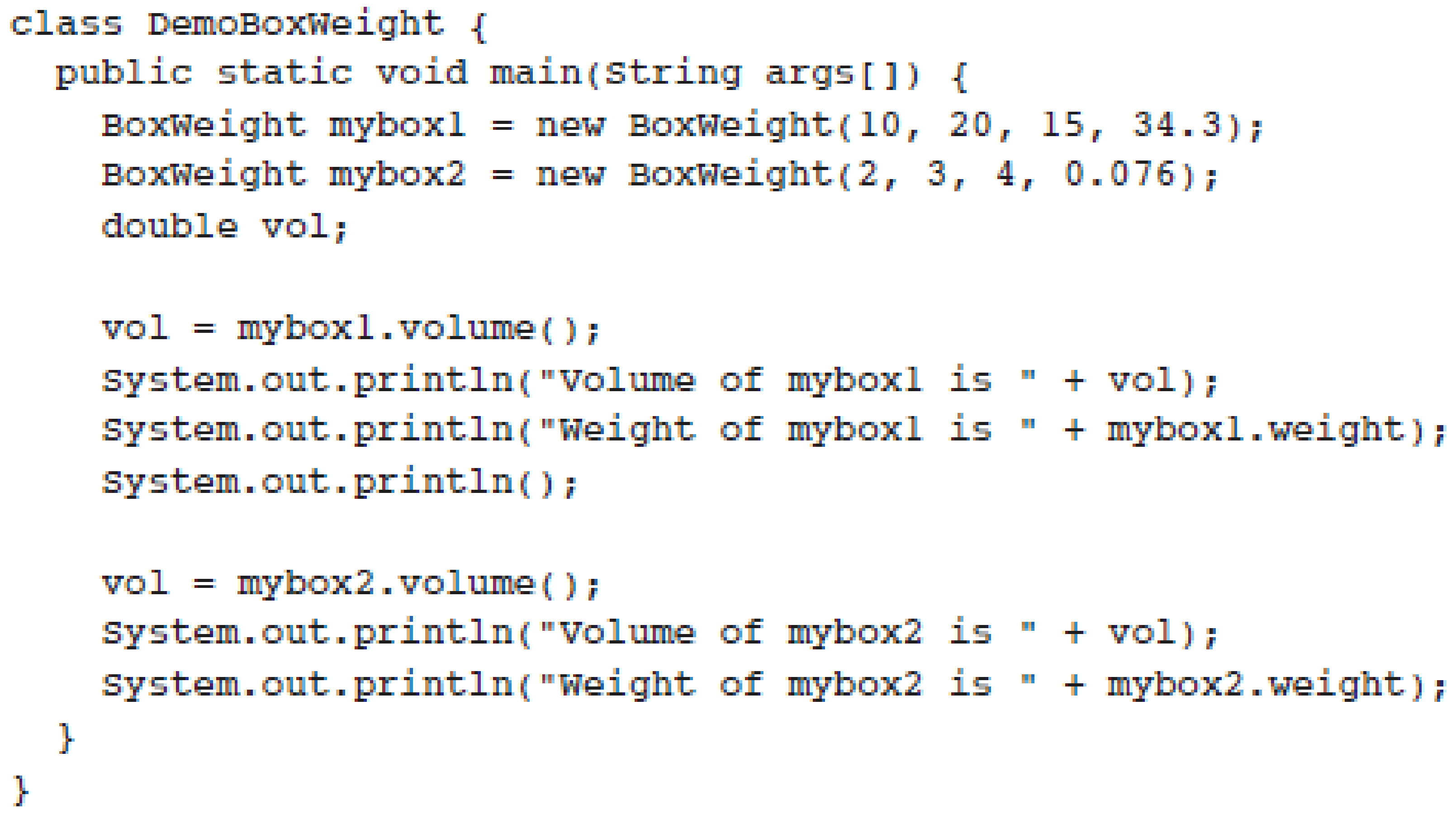
- BoxWeight class extends Box with the new weight variable:



```java
// Here, Box is extended to include weight.
class BoxWeight extends Box {
  double weight; // weight of box

  // constructor for BoxWeight
  BoxWeight(double w, double h, double d, double m) {
    width = w;
    height = h;
    depth = d;
    weight = m;
  }
}
```

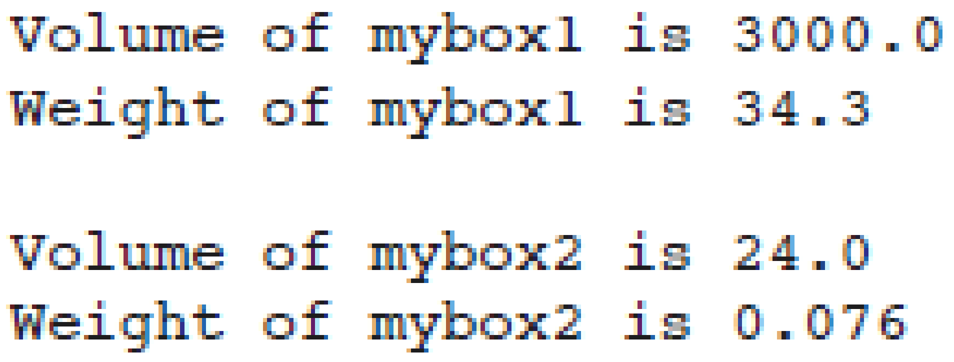
- Box is a super-class, BoxWeight is a sub-class.

Example: BoxWeightDemo



```
class DemoBoxWeight {
  public static void main(String args[]) {
    BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
    BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
    double vol;

    vol = mybox1.volume();
    System.out.println("Volume of mybox1 is " + vol);
    System.out.println("Weight of mybox1 is " + mybox1.weight);
    System.out.println();

    vol = mybox2.volume();
    System.out.println("Volume of mybox2 is " + vol);
    System.out.println("Weight of mybox2 is " + mybox2.weight);
  }
}
```

OUTPUT



```
Volume of mybox1 is 3000.0
Weight of mybox1 is 34.3

Volume of mybox2 is 24.0
Weight of mybox2 is 0.076
```

Another Sub-Class

Once a super-class exists that defines the attributes common to a set of objects, it can be used to create any number of more specific sub-classes.

The following sub-class of Box adds the color attribute instead of weight:

```
class ColorBox extends Box {
   int color;

   ColorBox(double w, double h, double d, int c) {
      width = w; height = h; depth = d;
      color = c;
   }
}
```

Referencing Sub-Class Objects

A variable of a super-class type may refer to any of its sub-class objects:

```
class SuperClass { … }
class SubClass extends SuperClass { … }

SuperClass o1;
SubClass o2 =  new SubClass();

o1 = o2;
```

However, the inverse is illegal:

~~o2 = o1~~

Example: Sub-Class Objects

```
class RefDemo {
  public static void main(String args[]) {
     BoxWeight weightbox = new BoxWeight(3, 5, 7, 8.37);
     Box plainbox = new Box(5, 5, 5);
     double vol;
     vol = weightbox.volume();
     System.out.print("Volume of weightbox is ");
     System.out.println(vol);
     System.out.print("Weight of weightbox is ");
     System.out.println(weightbox.weight);
     plainbox = weightbox;
     vol = plainbox.volume();
     System.out.println("Volume of plainbox is " + vol);
  }
}
```

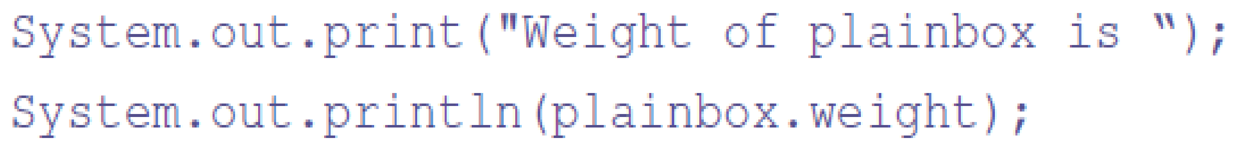
Super-Class Variable Access

plainbox variable now refers to the WeightBox object.

Can we then access this object’s weight variable through plainbox?

No. The type of a variable, not the object this variable refers to, determines which members we can access!

This is illegal:



```
System.out.print("Weight of plainbox is ");
System.out.println(plainbox.weight);
```

Super as Constructor

Calling a constructor of a super-class from the constructor of a sub-class:

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

Must occur as the very first instructor in the sub-class constructor:

```
class SuperClass { … }

class SubClass extends SuperClass {
   SubClass(…) {
       super(…);
       …
   }
   …
}
```

Example: Super Constructor

BoxWeight need not initialize the variable for the Box super-class, only the added weight variable:

```
class BoxWeight extends Box {
  double weight;

  BoxWeight(double w, double h, double d, double m) {
     super(w, h, d); weight = m;
  }

  BoxWeight(Box b, double w) {
     super(b); weight = w;
  }
}
```

Example: Super Constructor

```
class DemoSuper {
  public static void main(String args[]) {
     BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
     BoxWeight mybox2 = new BoxWeight(mybox1, 10.5);
     double vol;
     vol = mybox1.volume();
     System.out.println("Volume of mybox1 is " + vol);
     System.out.print("Weight of mybox1 is “);
     System.out.println(mybox1.weight);
     vol = mybox2.volume();
     System.out.println("Volume of mybox2 is " + vol);
     System.out.print("Weight of mybox2 is “);
     System.out.println(mybox2.weight);
  }
}
```

Referencing Sub-Class Objects

Sending a sub-class object:

```
BoxWeight mybox1 = new BoxWeight(10, 20, 15, 34.3);
BoxWeight mybox2 = new BoxWeight(mybox1, 10.5);
```

to the constructor expecting a super-class object:

```
BoxWeight(Box b, double w) {
   super(b); weight = w;
}
```

Uses of Super

Two uses of super:

- to invoke the super-class constructor
  - super();
- to access super-class members
  - super.variable;
  - super.method(…);

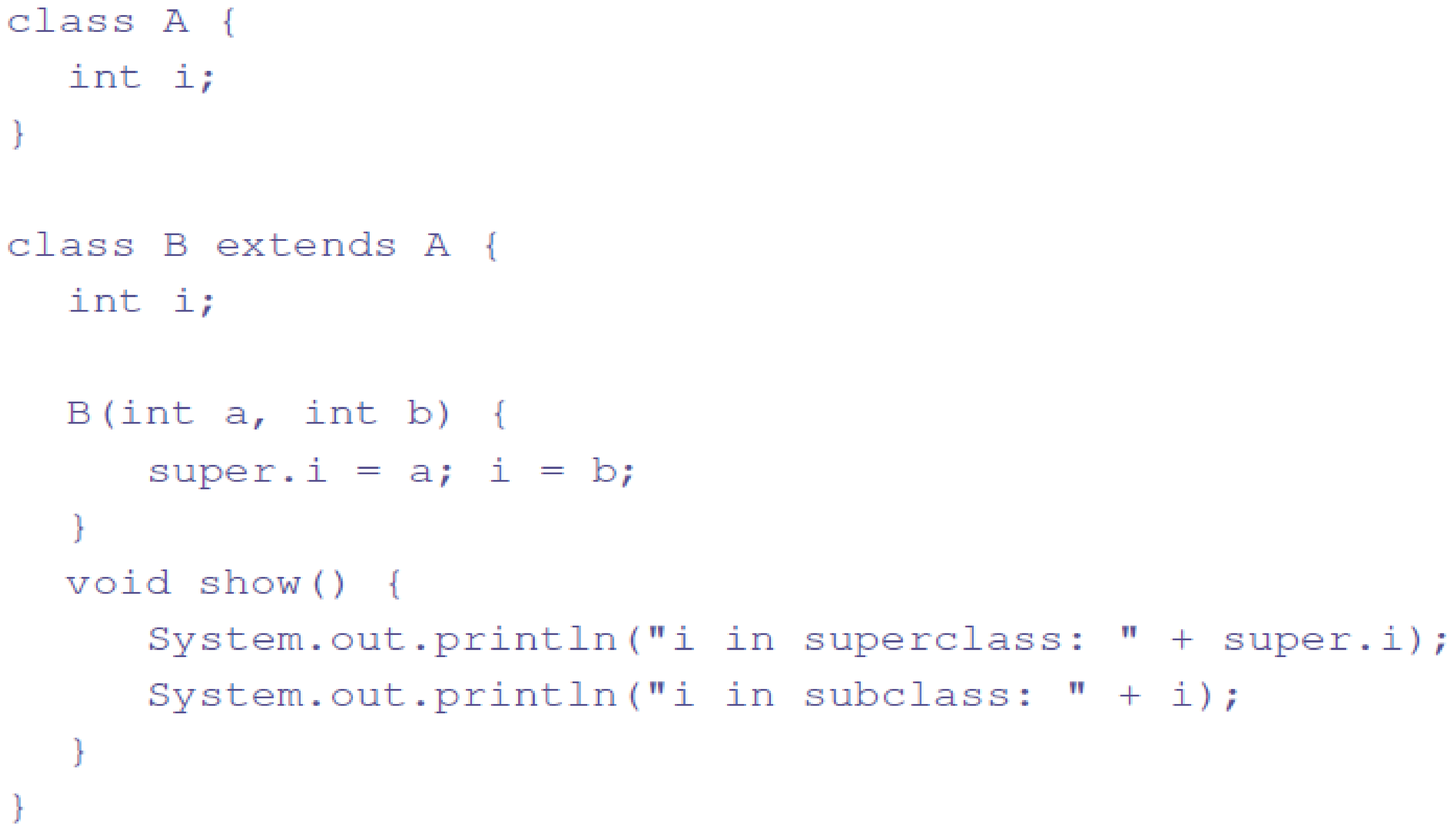
Super and Hiding

- Why is super needed to access super-class members?
- When a sub-class declares the variables or methods with the same names and types as its super-class:

```
class A {
   int i = 1;
}

class B extends A {
   int i = 2;
   System.out.println("i is " + i);
}
```

- The re-declared variables/methods hide those of the super-class.

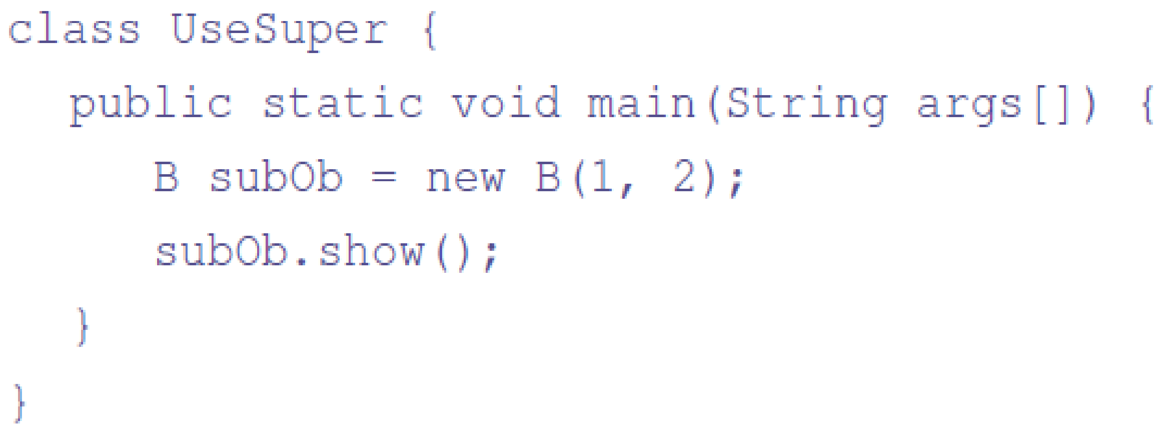
Example: Super and Hiding



```
class A {
  int i;
}

class B extends A {
  int i;

  B(int a, int b) {
     super.i = a; i = b;
  }
  void show() {
     System.out.println("i in superclass: " + super.i);
     System.out.println("i in subclass: " + i);
  }
}
```

Example: Super and Hiding

Although the i variable in B hides the i variable in A, super allows access to the hidden variable of the super-class:



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

Multi-Level Class Hierarchy

The basic Box class:

```
class Box {
  private double width, height, depth;
  Box(double w, double h, double d) {
     width = w; height = h; depth = d;
  }
  Box(Box ob) {
     width = ob.width;
     height = ob.height; depth = ob.depth;
  }
  double volume() {
     return width * height * depth;
  }
}
```

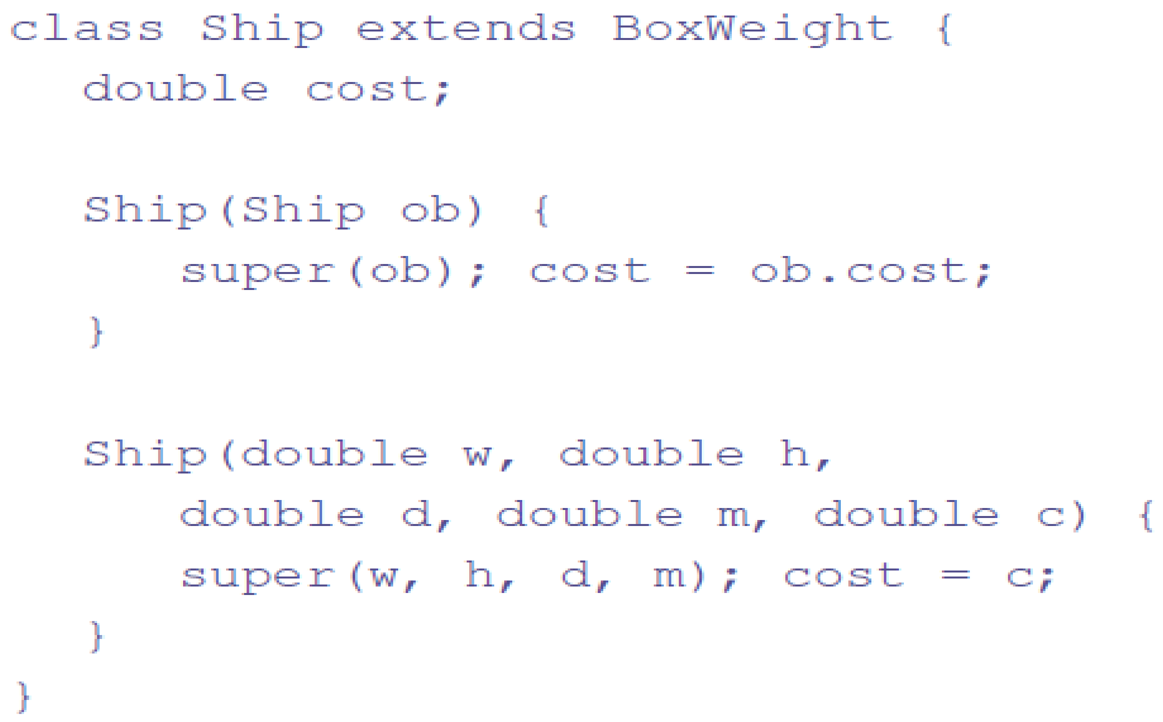
Multi-Level Class Hierarchy

Adding the weight variable to the Box class:

```
class BoxWeight extends Box {
  double weight;

  BoxWeight(BoxWeight ob) {
     super(ob); weight = ob.weight;
  }

  BoxWeight(double w, double h, double d, double m) {
     super(w, h, d); weight = m;
  }

}
```

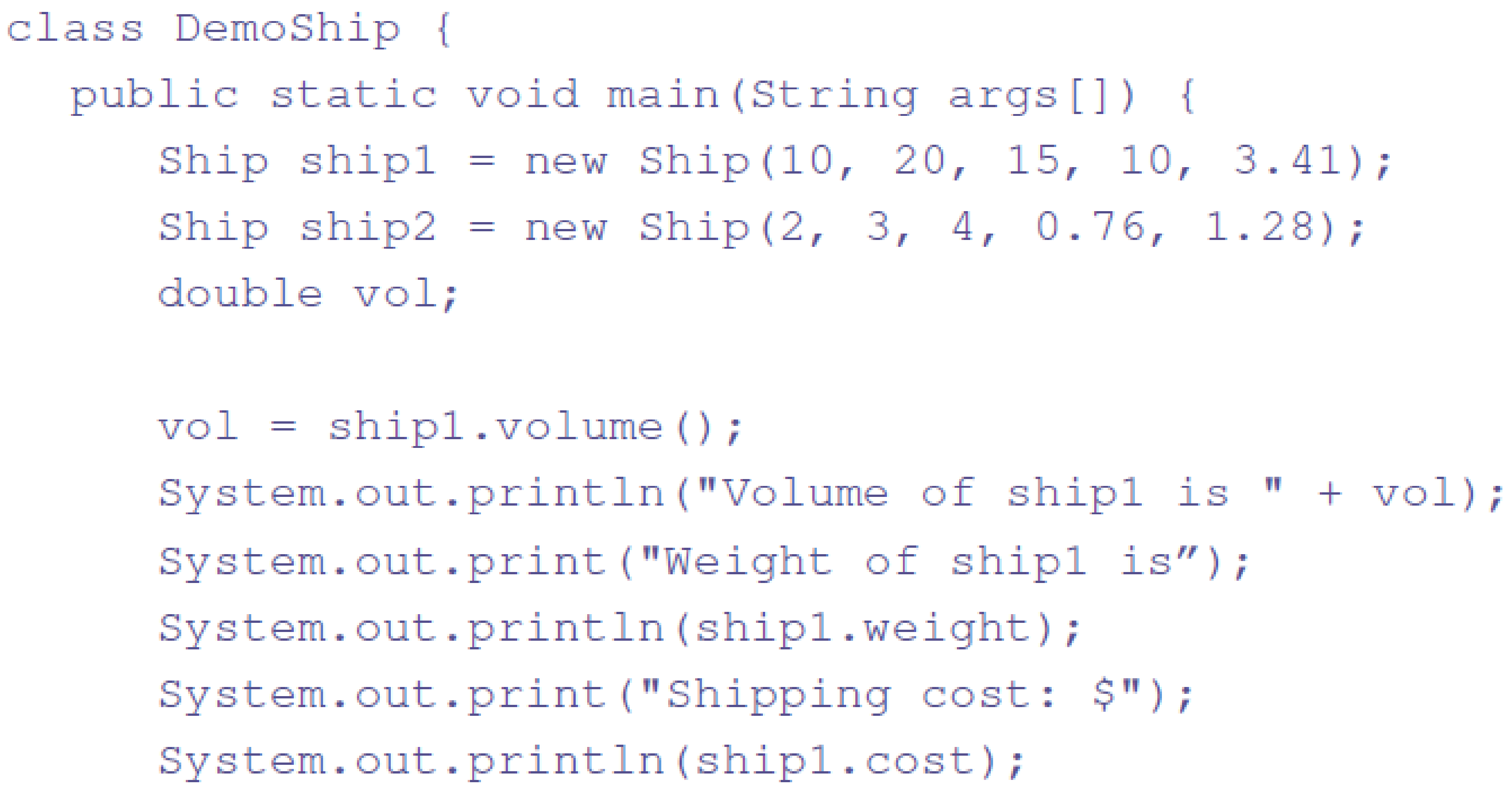
Multi-Level Class Hierarchy

Adding the cost variable to the BoxWeight class:



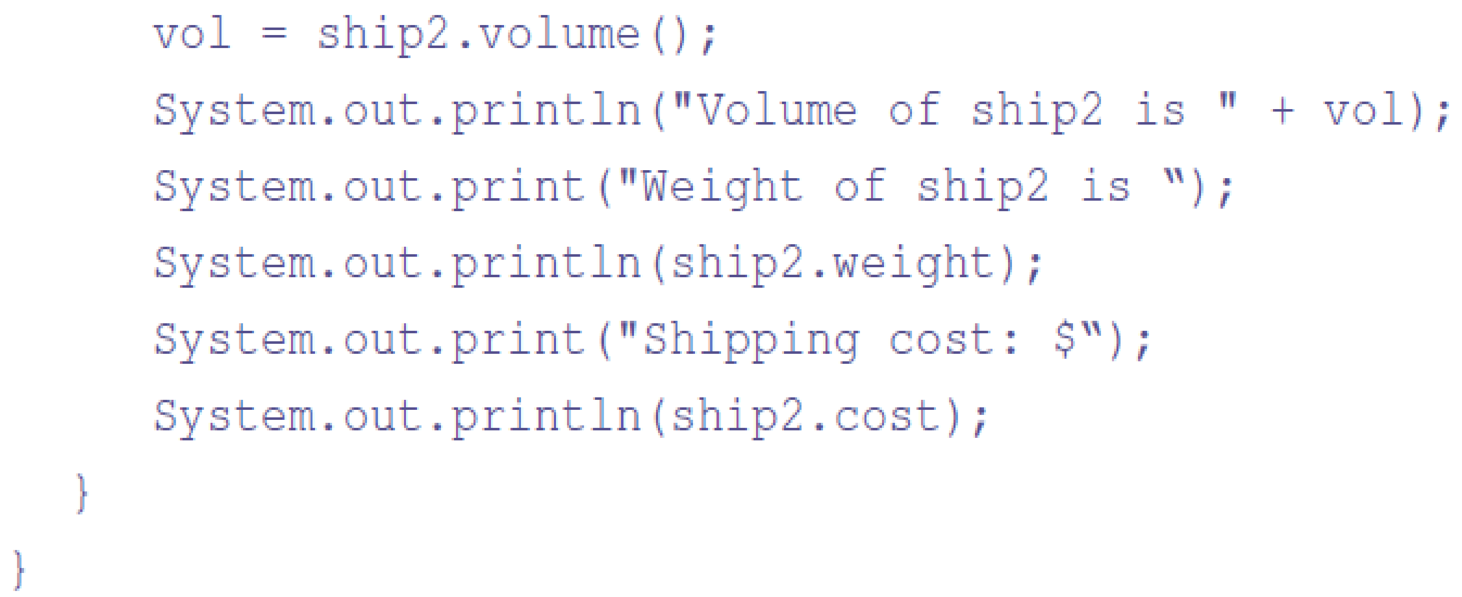
```
class Ship extends BoxWeight {
   double cost;

   Ship(Ship ob) {
      super(ob); cost = ob.cost;
   }

   Ship(double w, double h,
      double d, double m, double c) {
      super(w, h, d, m); cost = c;
   }
}
```

Multi-Level Class Hierarchy



```
class DemoShip {
  public static void main(String args[]) {
     Ship ship1 = new Ship(10, 20, 15, 10, 3.41);
     Ship ship2 = new Ship(2, 3, 4, 0.76, 1.28);
     double vol;

     vol = ship1.volume();
     System.out.println("Volume of ship1 is " + vol);
     System.out.print("Weight of ship1 is”);
     System.out.println(ship1.weight);
     System.out.print("Shipping cost: $");
     System.out.println(ship1.cost);
```

Multi-Level Class Hierarchy



```
    vol = ship2.volume();
    System.out.println("Volume of ship2 is " + vol);
    System.out.print("Weight of ship2 is ");
    System.out.println(ship2.weight);
    System.out.print("Shipping cost: $");
    System.out.println(ship2.cost);
  }
}
```

Constructor Call-Order

Constructor call-order:

- first call super-class constructors
- then call sub-class constructors

In the sub-class constructor, if super(…) is not used explicitly, Java calls the default, parameter-less super-class constructor.

Example: Constructor Call-Order

A is the super-class:

```
class A {
  A() {
     System.out.println("Inside A's constructor.");
  }
}
```

B and C are sub-classes of A:

```
class B extends A {
  B() {
     System.out.println("Inside B's constructor.");
  }
}
```

Example: Constructor Call-Order

```
class C extends B {
  C() {
     System.out.println("Inside C's constructor.");
  }
}
```

CallingCons creates a single object of the class C:

```
class CallingCons {
   public static void main(String args[]) {
      C c = new C();
   }
}
```

Class Participation

```
class A{
A()
{System.out.println("A");
}
A(int a)
{System.out.println("a");
}}
```

```
class B extends A
{
B(int a)
{super(a);
System.out.println("B");
}
}
```

```
class hello
{
public static void main(String abc[])
{
B b1=new B(5);}
}
```

Class Participation

```
class A{
A()
{System.out.println("A");
}
A(int a)
{System.out.println("a");
}}
```

```
class B extends A
{
B(int a)
{System.out.println("B");
}
}
```

```
class hello
{
public static void main(String abc[])
{
B b1=new B(5);}
}
```

Class Participation

```
class A{
A(int a)
{System.out.println("a");
}}
```

```
class B extends A
{
B(int a)
{System.out.println("B");
}
}
```

```
class hello
{
public static void main(String abc[])
{
B b1=new B(5);}
}
```

```
Exception in thread "main" java.lang.Error: Unresolved
compilation problem:
Implicit super constructor A() is undefined. Must
explicitly invoke another constructor
```

Class Participation

```
class A{
A(int a)
{System.out.println("a");
}}
```

```
class B extends A
{
B(int a)
{super(a);
System.out.println("B");
}
}
```

```
class hello
{
public static void main(String abc[])
{
B b1=new B(5);}
}
```

Task

Given a string s, the task is to check if it is palindrome or not.

Example:

Input: s = “abba”

Output: 1

Explanation: s is a palindrome

Input: s = “abc”

Output: 0

Explanation: s is not a palindrome

Class Participation

```
String str = "Hello";

str.concat(" World");

System.out.println(str);
```

Class Participation

Write a program that counts the number of vowels and consonants in a given string.

Input: "Java Programming"

Output: Vowels: 5, Consonants: 10

Modify the withdraw() method so that the account balance is not allowed to go below zero.

```
class BankAccount {
    private double balance;
    public BankAccount(double balance) {
        this.balance = balance;    }
    public void withdraw(double amount) {
        // Fill in the missing code to check if withdrawal is possible
        // If balance is sufficient, deduct amount
        // Otherwise, print "Insufficient funds"}
    public double getBalance() {
        return balance;    }

    public static void main(String[] args) {
      BankAccount account = new BankAccount(1000);
      account.withdraw(1200); // Should print "Insufficient funds"
      System.out.println("Remaining Balance: " + account.getBalance());    } }
```

Class Participation

Which loop is best suited for iterating over an array in Java?

a) for loop

b) while loop

c) for-each loop

d) Both a) and c)

Class Participation

How do you copy an array in Java?

a) int[] newArr = arr;

b) int[] newArr = Arrays.copyOf(arr, arr.length);

c) int[] newArr = arr.clone();

d) Both b) and c)

Class Participation

Which of the following is used to sort an array in Java?

a) Collections.sort()

b) Arrays.sort()

c) sortArray()

d) arr.sort()

Questions?