faheem Akhtar Rajputt

Object Oriented Programing (JAVA)

Lecture 17

Recap

- What is Inheritance
- Keyword *extends*
- Super and subclass
- Member access
- Extension as an advantage

Multilevel Hierarchy

- You can build hierarchies that contain as many layers of inheritance
- It is perfectly acceptable to use a subclass as a superclass of another

A, **B**, and **C**

Where,

- **C** can be a subclass of **B**,
- And **B** is a subclass of **A**

```
class Box {
  private double width;
  private double height;
  private 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;
  }
}
```

```
// Add weight.
class BoxWeight extends Box {
  double weight; // weight of box

  // construct clone of an object
  BoxWeight(BoxWeight ob) { // pass object to constructor
    super(ob);
    weight = ob.weight;
  }

  // constructor when all parameters are specified
  BoxWeight(double w, double h, double d, double m) {
    super(w, h, d); // call superclass constructor
    weight = m;
  }

  // default constructor
  BoxWeight() {
    super();
    weight = -1;
  }

  // constructor used when cube is created
  BoxWeight(double len, double m) {
    super(len);
    weight = m;
  }
}
```

```
// Add shipping costs
class Shipment extends BoxWeight {
  double cost;

  // construct clone of an object
  Shipment(Shipment ob) { // pass object to constructor
    super(ob);
    cost = ob.cost;
  }

  // constructor when all parameters are specified
  Shipment(double w, double h, double d,
           double m, double c) {
    super(w, h, d, m); // call superclass constructor
    cost = c;
  }

  // default constructor
  Shipment() {
    super();
    cost = -1;
  }

  // constructor used when cube is created
  Shipment(double len, double m, double c) {
    super(len, m);
    cost = c;
  }
}
```

OUTPUT

```
Volume of shipment1 is 3000.0
Weight of shipment1 is 10.0
Shipping cost: $3.41
Volume of shipment2 is 24.0
Weight of shipment2 is 0.76
Shipping cost: $1.28
```

```
class DemoShipment {
  public static void main(String args[]) {
    Shipment shipment1 =
           new Shipment(10, 20, 15, 10, 3.41);
    Shipment shipment2 =
           new Shipment(2, 3, 4, 0.76, 1.28);

    double vol;

    vol = shipment1.volume();
    System.out.println("Volume of shipment1 is " + vol);
    System.out.println("Weight of shipment1 is "
                       + shipment1.weight);
    System.out.println("Shipping cost: $" + shipment1.cost);
    System.out.println();

    vol = shipment2.volume();
    System.out.println("Volume of shipment2 is " + vol);
    System.out.println("Weight of shipment2 is "
                       + shipment2.weight);
    System.out.println("Shipping cost: $" + shipment2.cost);
  }
}
```

Constructors Call

- When a class hierarchy is created, in what order are the constructors for the classes that make up the hierarchy called?
- For example, given a subclass called B and a superclass called A, is A's constructor called before B's, or vice versa?
- The answer is that in a class hierarchy, constructors are called in order of derivation.

When Constructors are called-Example

OUTPUT
Inside A's constructor
Inside B's constructor
Inside C's constructor

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

```
// Create another subclass by extending B.
class C extends B {
  C() {
    System.out.println("Inside C's
constructor.");
  }
}
```

```
//Create a subclass by extending class A.
class B extends A {
  B() {
    System.out.println("Inside B's
constructor.");
  }
}
```

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

Method Overriding

- In a class hierarchy, when a method in a subclass has the same name and type signature as a method in its superclass that is called method overriding
- So, when an overridden method is called from within a subclass, it will always refer to the version of that method defined by the subclass.
- The version of the method defined by the superclass will be hidden

Method Overriding

OUTPUT
k: 3

```
class A {
  int i, j;

  A(int a, int b) {
    i = a;
    j = b;
  }

  // display i and j
  void show() {
    System.out.println("i and j: " + i + " " + j);
  }
}
```

```
class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }

  // display k -- this overrides show() in A
  void show() {
    System.out.println("k: " + k);
  }
}
```

```
class Override {
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);

    subOb.show(); // this calls show() in B
  }
}
```

SO...

How to call superclass OVERRIDEN method

?

Here is the answer…

```
class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }

  void show() {
    super.show(); // this calls A's show()
    System.out.println("k: " + k);
  }
}
```

OUTPUT

i and j: 1 2
k: 3

Method Overloading

- Method overriding occurs only when the names and the type signatures of the two methods are identical.
- If they are not, then the two methods are simply overloaded.

Method Overloading

```
/* Methods with differing type signatures are
overloaded -- not overridden.*/
class A {
  int i, j;

  A(int a, int b) {
    i = a;
    j = b;
  }

  // display i and j
  void show() {
    System.out.println("i and j: " + i + " " + j);
  }
}
```

```
// Create a subclass by extending class A.
class B extends A {
  int k;

  B(int a, int b, int c) {
    super(a, b);
    k = c;
  }

  // overload show()
  void show(String msg) {
    System.out.println(msg + k);
  }
}
```

```
class OverLoading{
  public static void main(String args[]) {
    B subOb = new B(1, 2, 3);

    subOb.show("This is k: "); // this calls show() in B
    subOb.show(); // this calls show() in A
  }
}
```

OUTPUT

This is k: 3
i and j: 1 2

Using Abstract Classes

- If you will want to create a superclass that only defines a generalized form that will be shared by all of its subclasses
- leaving it to each subclass to fill in the details.
- class determines the nature of the methods that the subclasses must implement.
- abstract keyword

Using Abstract Classes

- Any class that contains one or more abstract methods must also be declared abstract.

```
abstract class A{  /* Statements */ }
```

- The methods in abstract class must be abstract also

```
abstract type name(parameter-list);
```

Using Abstract Classes

```
// A Simple demonstration of abstract.
abstract class A {
  abstract void callme();

  // concrete methods are still allowed in abstract classes
  void callmetoo() {
    System.out.println("This is a concrete method.");
  }
}
```

```
class B extends A {
  void callme() {
    System.out.println("B's implementation of callme.");
  }
}
```

```
class AbstractDemo {
  public static void main(String args[]) {
    B b = new B();

    b.callme();
    b.callmetoo();
  }
}
```

OUTPUT:

This is a concrete method.
B's implementation of callme.

Things to remember

- As mentioned, it is not possible to instantiate an abstract class by creating its object
- Class A implements a concrete method called callmetoo( ). It is perfectly fine.

3 usage of final

- First: already defined
- Second: it disallow a method from being overrideng.

```
final void meth() { /*Statements*/ }

void meth() { /*Statements*/ }    // Not allowed
```

- Third: Sometimes you also want to prevent your class from being inherited

```
final class A { /* Statements */ }
class B extends A { //...      // Not allowed
```

Chapter Review

- Inheritance basics
- Using super
- Creating Multilevel Hierarchy
- When Constructors are called
- Method overriding
- Using Abstract Classes
- Using Final with inheritance

Questions?


Sukkur
IBA
Thanks…