Object Oriented Programming

Inheritance

Inheritance

- Inheritance is one of the cornerstones of object-oriented programming because it allows the creation of hierarchical classifications
- Using inheritance, you can create a general class that defines traits common to a set of related items
- This class can then be inherited by other, more specific classes, each adding those things that are unique to it

Inheritance

- In the terminology of Java, a class that is inherited is called a superclass. The class that does the inheriting is called a subclass
- Therefore, a subclass is a specialized version of a superclass. It inherits all of the members defined by the superclass and adds its own, unique elements

Inheritance basics

- To inherit a class, you simply incorporate the definition of one class into another by using the extends keyword
- The following program creates a superclass called A and a subclass called B. Notice how the keyword extends is used to create a subclass of A.

Example

clipse IDE

Run Window Help

switch_1.java switchseason.java breakexample.java box.

```
1  class A{
2      int i, j;
3      void showij() {
4          System.out.println("i and j:"+i+" "+j);
5      }
6  }
7  class B extends A{
8      int k;
9      void showk() {
10         System.out.println("K:"+ k);
11     }
12     void sum() {
13         System.out.println("i+J+K"+(i+j+k));
14     }
15 }
```

Example

```
public class simpleinheritance {

    public static void main(String[] args) {
        A superob=new A();
        B subob=new B();
        superob.i=10;
        superob.j=20;
        System.out.println("Contents of superob");
        System.out.println();
        superob.showij();
        subob.i=49;
        subob.j=68;
        subob.k=93;
        System.out.println("Contents of subob");
        subob.showij();
        subob.showk();
        System.out.println();
        System.out.println("Sum of i,j, and k in subob");
        subob.sum();// TODO Auto-generated method stub

    }

}
```

Example output

```
<terminated> SimpleInheritance [Java Application] C:\Pr
Contents of superob

i and j:10 20
Contents of subob
i and j:49 68
K:93

Sum of i,j, and k in subob
i+J+K210
```

Method access and inheritance

- Although a subclass includes all of the members of its superclass, it cannot access those members of the superclass that have been declared as private
- For example, consider the following simple class hierarchy:

Example

```java
class A1{
    static int i;
    private int j;
    void setj( int j1) {
        j=j1;
    }
    int getj() {
        return j;
    }
}
class B1 extends A1{
    int sum() {
        return i+j;
    }
}
public class Access {

    public static void main(String[] args) {
        B1 ob=new B1();
        B1.i=49;
        B1.j=77;// TODO Auto-generated method stub
        System.out.println(B1.sum());
```

Console

```
<terminated> Access [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (Oct 28, 2019, 3:57:10 PM)
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The field A1.j is not visible
```

Example

it.java - Eclipse IDE

Search Project Run Window Help

Light.java | Scannerint.java | pwr.java | savingaccoun... | boxinherit.java | Boxdemo.java | Box.java | boxdemo2.java

```java
class box21{
    double height,width,depth;
    box21(box21 ob){
        ob.height=height;
        ob.width=width;
        ob.depth=depth;
    }
    box21(double h, double w, double d){
        height=h;
        width=w;
        depth=d;
    }
    box21(){
        height=-1;
        width=-1;
        depth=-1;
    }
    box21(double len ){
        height=width=depth=len;
    }
    double volume() {
        return height*width*depth;
    }

}
class boxweight extends box21{
    double weight;
    boxweight(double h1, double w1, double d1, double weigh)
{
        height=h1;
        width=w1;
        depth=d1;
        weight=weigh;
}
}
public class boxinherit {

    public static void main(String[] args) {
        boxweight box=new boxweight(48.23,28.5,20.6,55.6);
        boxweight box1=new boxweight(40.23,25.5,20.6,60.6);
        System.out.println("The volume of box is "+box.volume());
        System.out.println("The weight of box is "+box.weight);
        System.out.println("The volume of box1 is "+box1.volume());
        System.out.println("The weight of box1 is "+box1.weight);

    }

}
```

Problems | Javadoc | Declaration | Console

```
<terminated> boxinherit [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Oct 29, 2019, 12:07:50 PM)
The volume of box is 28315.833
The weight of box is 55.6
The volume of box1 is 21132.819000000003
The weight of box1 is 60.6
```

Using super

- There will be times when you will want to create a superclass that keeps the details of its implementation to itself (that is, that keeps its data members private)
- In this case, there would be no way for a subclass to directly access or initialize these variables on its own
- Since encapsulation is a primary attribute of OOP, it is not surprising that Java provides a solution to this problem
- Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword super.

Using super

- super has two general forms.
- The first calls the superclass' constructor
- The second is used to access a member of the superclass that has been hidden by a member of a subclass

Using super to call superclass constructors

- A subclass can call a constructor defined by its superclass by use of the following form of super:

  super(arg-list);

Here, arg-list specifies any arguments needed by the constructor in the superclass. super( ) must always be the first statement executed inside a subclass' constructor.

Example

/shapes4.java - Eclipse IDE

Navigate Search Project Run Window Help

Light.java | Scannerint.java | pwr.java | bank.java | charge.java | circle.java | p.java | student.java | bo

```java
class twodshape{
    private double width;
    private double height;
    twodshape(double w, double h){
        width=w;
        height=h;
    }
    void setwidth(double w) {
        width=w;
    }
    double getwidth() {
        return width;
    }
    void setheight(double h) {
        height=h;
    }
    double getheight() {
        return height;
    }
    void showdim() {
        System.out.println("width and height are"+width+ " and "+ height);
    }
}
class triangle3 extends twodshape{
    private String style;
    triangle3(String s, double w, double h){
        super(w,h);
        style=s;
    }
    double area() {
        return getwidth()*getheight()/2;
    }
    void showstyle() {
        System.out.println("Triangle is"+style);
    }
}
public class shapes4 {

    public static void main(String[] args) {
        triangle3 t1=new triangle3("filled", 4.0,4.0);
        triangle3 t2= new triangle3("outlined", 8.0, 12.0);// TODO Auto-generated method stub
        System.out.println("Info for t1:");
        t1.showdim();
        t1.showstyle();
        System.out.println("Area is"+t1.area());
        System.out.println();
        System.out.println("Info for t2:");
        t2.showdim();
        t2.showstyle();
        System.out.println("Area is"+t2.area());
    }
```

Problems | Javadoc | Declaration | Console

```
<terminated> shapes4 [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 2, 2019, 5:32:08 PM)
width and height are4.0 and 4.0
Triangle isfilled
Area is8.0

Info for t2:
width and height are8.0 and 12.0
```

Using super to access superclass members

- There is a second form of super that acts somewhat like this, except that it always refers to the superclass of the subclass in which it is used. This usage has the following general form:

  super.member

Here, member can be either a method or an instance variable.

Example

/usesuper.java - Eclipse IDE

Navigate Search Project Run Window Help

Light.java | Scannerint.java | pwr.java | bank.java | charge.java | circle.java | p.java | boxin

```java
class A{
    int i;
}
class B2 extends A{
    int i;
    B2(int a, int b){
        super.i=a;
        i=b;
    }
    void show() {
        System.out.println("i in superclass:" +super.i);
        System.out.println("i in superclass:" +i);
    }
}
public class usesuper {

    public static void main(String[] args) {
        B2 subob= new B2(1,2);
        subob.show();// TODO Auto-generated method stub

    }

}
```

Problems | Javadoc | Declaration | Console

```
<terminated> usesuper [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 2, 2019, 5:57:36 PM)
i in superclass:1
i in superclass:2
```

Example

usesuper.java - Eclipse IDE

avigate Search Project Run Window Help

Light.java | Scannerint.java | pwr.java | bank.java | charge.java | circle.java

```java
class A{
    private int i;
}
class B2 extends A{
    int i;
    B2(int a, int b){
        super.i=a;
        i=b;
    }
    void show() {
        System.out.println("i in superclass:" +super.i);
        System.out.println("i in superclass:" +i);
    }
}
public class usesuper {

    public static void main(String[] args) {
        B2 subob= new B2(1,2);
        subob.show();// TODO Auto-generated method stub

    }

}
```

Problems | Javadoc | Declaration | Console

```
<terminated> usesuper [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 2, 2019, 5:59:53 PM)
Exception in thread "main" java.lang.Error: Unresolved compilation problems:
        The field A.i is not visible
        The field A.i is not visible

        at B2.<init>(usesuper.java:7)
        at usesuper.main(usesuper.java:18)
```

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, then the method in the subclass is said to override the method in the superclass
- When an overridden method is called from within its 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

Example

src/methodoverride.java - Eclipse IDE

Navigate Search Project Run Window Help

Light.java | Scannerint.java | pwr.java | bank.java | charge.java | circle.java | boxinherit.java | shap

```java
class A2{
    int i,j;
    A2(int a, int b){
        i=a;
        j=b;
    }
    void show() {
        System.out.println("i and j:"+ i + " "+ j);
    }
}
class B1 extends A2{
    int k;
    B1(int a, int b, int c){
        super(a,b);
        k=c;
    }
    void show() {

        System.out.println("K:"+k);
    }
}
public class methodoverride {

    public static void main(String[] args) {
        B1 subob=new B1(1,2,3);
        subob.show();// TODO Auto-generated method stub

    }

}
```

Problems | Javadoc | Declaration | Console

```
<terminated> methodoverride [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 2, 2019, 6:18:40 PM)
K:3
```

Example

'src/methodoverride.java - Eclipse IDE

Navigate Search Project Run Window Help

Light.java | Scannerint.java | pwr.java | bank.java | charge.java | circle.java | boxinherit.java | shapes4.java

```java
class A2{
    int i,j;
    A2(int a, int b){
        i=a;
        j=b;
    }
    void show() {
        System.out.println("i and j:"+ i + " "+ j);
    }
}
class B1 extends A2{
    int k;
    B1(int a, int b, int c){
        super(a,b);
        k=c;
    }
    void show() {
        super.show();
        System.out.println("K:"+k);
    }
}
public class methodoverride {

    public static void main(String[] args) {
        B1 subob=new B1(1,2,3);
        subob.show();// TODO Auto-generated method stub

    }

}
```

Problems | Javadoc | Declaration | Console

```
<terminated> methodoverride [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 2, 2019, 6:17:30 PM)
i and j:1 2
K:3
```

Creating a multi level hierarchy

- you can build hierarchies that contain as many layers of inheritance as you like
- As mentioned, it is perfectly acceptable to use a subclass as a superclass of another For example, given three classes called A, B, and C, C can be a subclass of B, which is a subclass of A. When this type of situation occurs, each subclass inherits all of the traits found in all of its superclasses. In this case, C inherits all aspects of B and A.

Multilevel hierarchy Example

src/demo/Shipment.java - Eclipse IDE

Navigate Search Project Run Window Help

Light.java | Scannerint.java | pwr.java | bank.java | charge.java | boxinherit.java | shapes4.java | usesuper.java | methodoverri... | *der

```java
class Box5{
    private double height;
    private double width;
    private double depth;
    Box5(double h, double w, double d){
        height=h;
        width=w;
        depth=d;
    }
    Box5(){
        height=-1;
        width=-1;
        depth=-1;
    }
    Box5(double len){
        height=width=depth=len;
    }
    double volume() {
        return height*width*depth;
    }
}
```

Multilevel hierarchy example

```
class boxweight3 extends Box5{
    private double weight;
    boxweight3(double h1, double w1, double d1, double m){
        super(h1,w1,d1);
        weight=m;
    }
    boxweight3(){
        super();
        weight=-1;
    }
    boxweight3(double len, double m){
        super(len);
        weight=m;
    }
    double getweight() {
        return weight;
    }
}
class shipment extends boxweight3{
```

Multilevel hierarchy example

```
class shipment extends boxweight3{
    double cost;
    shipment(double h2, double w2, double d2, double m1, double c){
        super(h2,w2,d2,m1);
        cost=c;
    }
    shipment(){
        super();
        cost=-1;
    }
    shipment(double len, double m1, double c){
        super(len,m1);
        cost=c;
    }
}
```

Multilevel hierarchy example

src/demoshipment.java - Eclipse IDE

Navigate Search Project Run Window Help

Light.java | Scannerint.java | pwr.java | bank.java | charge.java | boxinherit.java | shapes4.java | usesuper.java

```
52          cost=c;
53      }
54  }
55  public class demoshipment {
56
57      public static void main(String[] args) {
58          shipment sh1= new shipment(48.5,20.5, 15.6, 50.5, 33.5);
59          shipment sh2=new shipment();
60          shipment sh3= new shipment(7, 58.3, 40.5);
61          System.out.println("Volume for sh1 is "+sh1.volume());
62          System.out.println("weight for sh1 is "+sh1.getweight());
63          System.out.println("cost for sh1 is "+sh1.cost);
64          System.out.println("Volume for sh2 is "+sh2.volume());
65          System.out.println("weight for sh2 is "+sh2.getweight());
66          System.out.println("cost for sh2 is "+sh2.cost);
67          System.out.println("Volume for sh3 is "+sh3.volume());
68          System.out.println("weight for sh3 is "+sh3.getweight());
69          System.out.println("cost for sh3 is "+sh3.cost);
70          // TODO Auto-generated method stub
71
72      }
73
74  }
75
```

Problems | Javadoc | Declaration | Console

```
<terminated> demoshipment [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 2, 2019, 6:59:24 PM)
Volume for sh1 is 15510.3
weight for sh1 is 50.5
cost for sh1 is 33.5
Volume for sh2 is -1.0
weight for sh2 is -1.0
cost for sh2 is -1.0
Volume for sh3 is 343.0
weight for sh3 is 58.3
cost for sh3 is 40.5
```

Method overriding without using super-dynamic method dispatch

- Method overriding forms the basis for one of Java’s most powerful

concepts: dynamic method dispatch

- When an overridden method is called through a superclass reference, Java determines which version of that method to execute based upon the type of the object being referred to at the time the call occurs. This is called dynamic method dispatch. In dynamic method dispatch, method overriding is resolved at run time rather than compile time.

Dynamic method dispatch

Window Help

Light.java | Scannerint.java | pwr.java | bank.java | charge.java | circle.java | demoshipmen...

```java
class figure{
    private double dim1;
    private double dim2;
    figure(double a, double b){
      dim1=a;
      dim2=b;
    }
    double getdim1() {
        return dim1;
    }
    double getdim2() {
        return dim2;
    }
    double area() {
        System.out.println("Areas for figure is undefined");
        return 0;
    }
}
```

Dynamic method dispatch

```
}
class rectangle3 extends figure{
    rectangle3(double a, double b){
        super(a,b);
    }
    double area() {
        System.out.println("Inside area for rectangle");
        return getdim1()*getdim2();
    }

}
class triangle4 extends rectangle3{
    triangle4(double a1, double b1){
        super(a1,b1);
    }
    double area() {
        System.out.println("Inside Area for triangle");
        return getdim1()*getdim2()/2;
    }

}
```

Dynamic method dispatch

```java
public class findareas {

    public static void main(String[] args) {
        figure f= new figure(10,10);// TODO Auto-generated method stub
        rectangle3 r= new rectangle3(9,5);
        triangle4 t= new triangle4(10,8);
        figure figref;
        figref=r;
        System.out.println("Area is"+figref.area());
        figref = t;
        System.out.println("Area is"+figref.area());
        figref=f;
        System.out.println("Area is"+figref.area());
    }

}
```

Problems @ Javadoc Declaration Console

```
<terminated> findareas [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 4, 2019, 7:15:27 PM)
Inside area for rectangle
Area is45.0
Inside Area for triangle
Area is40.0
Areas for figure is undefined
Area is0.0
```

abstract classes

- There are situations in which you will want to define a superclass that declares the structure of a given abstraction without providing a complete implementation of every method
- That is, sometimes 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
- You can require that certain methods be overridden by subclasses by

specifying the abstract type modifier

abstract classes

- A class which is declared as abstract is known as an abstract class. It can have abstract and non-abstract methods. It needs to be extended and its method implemented. It cannot be instantiated.
- To declare an abstract method, use this general form:

  abstract type name(parameter-list)

abstract class example

un Window Help

Light.java | Scannerint.java | pwr.java | bank.java | demoshipmen... | javaimage.java | find

```java
abstract class A5{
    abstract void callme();
    void callmetoo() {
        System.out.println("This is a concrete method");
    }
}
class B5 extends A5{
    void callme() {
        System.out.println("B's implementtation of callme.");
    }
}
public class abstractdemo {

    public static void main(String[] args) {
        //A a = new A();
        B5 b= new B5();
        b.callme();
        b.callmetoo();// TODO Auto-generated method stub

    }

}
```

Problems | Javadoc | Declaration | Console

```
<terminated> abstractdemo [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 4, 2019, 9:23:41 PM)
B's implementtation of callme.
This is a concrete method
```

abstract class example

E
un Window Help

Light.java | Scannerint.java | demoshipmen... | javaimage.java | findareas.java | abstractdem... | finalmethod.... | finalin

```java
abstract class A5{
    abstract void callme() {
        System.out.println("Implementation");
    }
    void callmetoo() {
        System.out.println("This is a concrete method");
    }
}
class B5 extends A5{
    void callme() {
        System.out.println("B's implementtation of callme.");
    }
}
public class abstractdemo {

    public static void main(String[] args) {
        A a = new A();
        //B5 b= new B5();
        a.callme();
        a.callmetoo();// TODO Auto-generated method stub

    }

}
```

Problems | Javadoc | Declaration | Console

5 errors, 13 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (5 items)				
Abstract methods do not specify a body	abstractdemo...	/ICE107/src	line 2	Java Problem

abstract class example

n Window Help

demoshipmen... javaimage.java findareas.java abstractdem... finalmethod.... finalinheri...

```java
abstract class emp10{
    private String firstname;
    private String lastname;
    private String socialsecuritynumber;
    emp10(String f, String l, String ssn){
        this.firstname=f;
        this.lastname=l;
        this.socialsecuritynumber=ssn;
    }
    String getfirstname() {
        return firstname;
    }
    String getlastname() {
        return lastname;
    }
    String getssn() {
        return socialsecuritynumber;
    }
    abstract double weeklysalary();
}
class salary extends emp10{
    double hour, rate;
    salary(String f1, String l1, String SSN, double h, double r){
        super(f1,l1,SSN);
        hour=h;
        rate=r;
    }
    double weeklysalary() {
        double overtime;
        if(hour>40) {
            overtime=hour-40;
            hour=hour-overtime;
            return (hour*rate)+overtime*rate*1.5;
        }
        else {
            return hour*rate;
        }
    }

}
```

abstract class example

```
41 public class employe {
42
43     public static void main(String[] args) {
44         //emp10 E= new emp10("Rafsun", "Islam", "445-550-220");// TODO Auto-ge
45           salary S= new salary("Rafsun", "Islam", "445-550-220",45, 10);
46           System.out.println("first Name is"+S.getfirstname());
47           System.out.println("Last name is"+S.getlastname());
48           System.out.println("Social security number is"+S.getssn());
49           System.out.println("Weekly salary is"+S.weeklysalary());
50     }
51
52 }
53
```

Problems @ Javadoc Declaration Console

```
<terminated> employe [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 4, 2019, 10:14
first Name isRafsun
Last name isIslam
Social security number is445-550-220
Weekly salary is475.0
```

Using final to prevent method overriding

- While method overriding is one of Java’s most powerful features, there will be times when you will want to prevent it from occurring
- To disallow a method from being overridden, specify final as a modifier at the start of its declaration
- Methods declared as final cannot be overridden

example

```java
class A6{
    final void callme() {
      System.out.println("This is A method");
}
}
class B6 extends A6{

void callme() {
        System.out.println("This is B method");
    }
}
public class finalmethod {

    public static void main(String[] args) {
        B6 b=new B6();
        b.callme();// TODO Auto-generated method stub

    }

}
```

Problems @ Javadoc Declaration Console

```
<terminated> finalmethod [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 4, 2019, 9:50:17 PM)
Exception in thread "main" java.lang.VerifyError: class B6 overrides final method callme.()V
        at java.lang.ClassLoader.defineClass1(Native Method)
        at java.lang.ClassLoader.defineClass(Unknown Source)
        at java.security.SecureClassLoader.defineClass(Unknown Source)
        at java.net.URLClassLoader.defineClass(Unknown Source)
        at java.net.URLClassLoader.access$100(Unknown Source)
        at java.net.URLClassLoader$1.run(Unknown Source)
        at java.net.URLClassLoader$1.run(Unknown Source)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.net.URLClassLoader.findClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at sun.misc.Launcher$AppClassLoader.loadClass(Unknown Source)
        at java.lang.ClassLoader.loadClass(Unknown Source)
        at finalmethod.main(finalmethod.java:15)
```

Using final to prevent inheritance

- Sometimes you will want to prevent a class from being inherited. To do this, precede the class declaration with final. Declaring a class as final implicitly declares all of its methods as final, too

Example

```java
final class A7{
    void callme() {
      System.out.println("This is A method");
}
}
class B7 extends A7{

void callme() {
        System.out.println("This is B method");
    }
}
public class finalinheritance {
public static void main(String[]args) {
    B7 b=new B7();
    b.callme();
}
}
```

```
<terminated> finalinheritance [Java Application] C:\Program Files\Java\jre1.8.0_161\bin\javaw.exe (Nov 4, 2019, 9:52:36 PM)
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The type B7 cannot subclass the final class A7

        at B7.<init>(finalinheritance.java:6)
        at finalinheritance.main(finalinheritance.java:14)
```