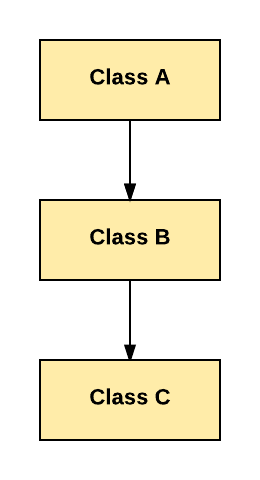
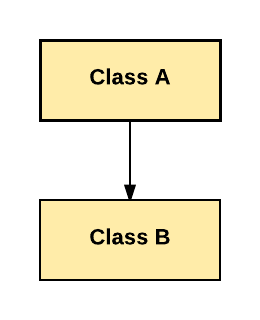
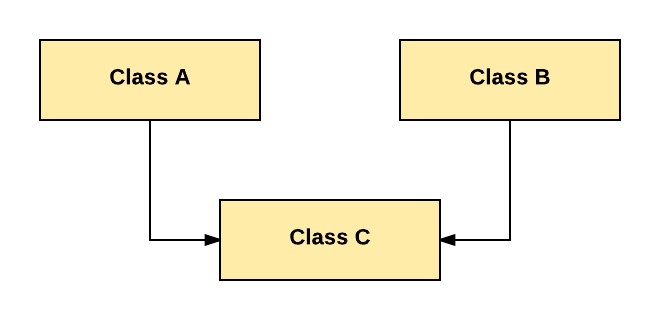
Inheritance

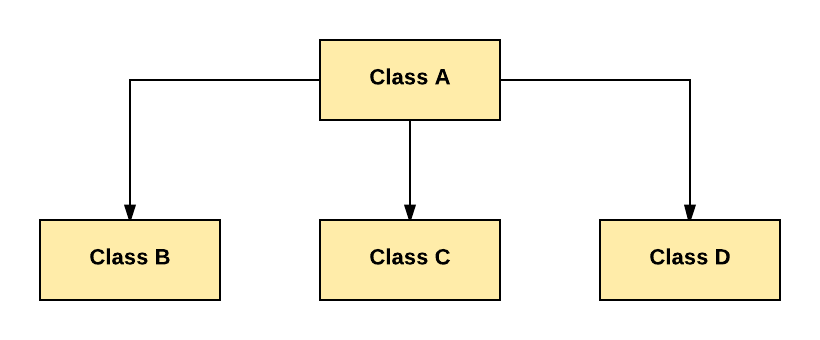
Types of inheritance

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hierarchical inheritance
5. Hybrid inheritance

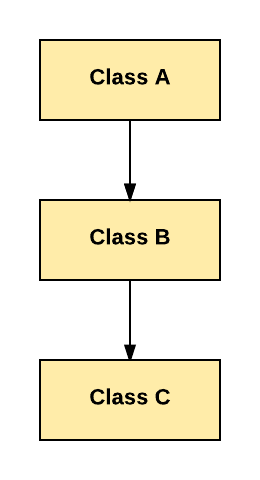
single inheritance

Multiple inheritance

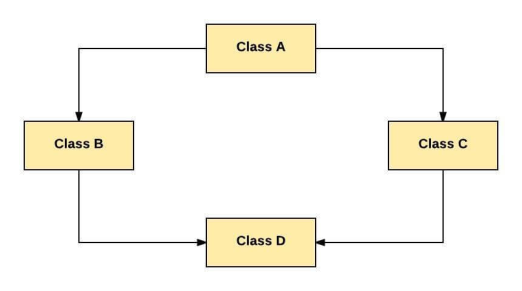


Hierarchical inheritance

Multilevel inheritance



Hybrid inheritance



* Java inheritance syntax

class subClass extends superClass

{

//methods and fields

}

* **Usage of java super Keyword**
* super can be used to refer immediate parent class instance variable.
* super can be used to invoke immediate parent class method.
* super() can be used to invoke immediate parent class constructor

Simple Inheritance example

|  |
| --- |
| **class Animal**  **{**  **void eat()**  **{**  **System.*out.println("eating");***  **}**  **}**  **class Dog extends Animal**  **{**  **void bark()**  **{**  **System.*out.println("barking");***  **}**  **}**  **public class simpleinherit {**  **public static void main(String[] args) {**  **{**  **Dog obj=new Dog();**  **obj.eat();**  **obj.bark();**  **}**  **}**  **}** |

Another example

|  |
| --- |
| class Box  {  double width;  double height;  double depth;  //constructor  Box(Box obj)  {  width=obj.width;  height=obj.height;  depth=obj.depth;  }  //constructor  Box(double w, double h,double d)  {  width=w;  height=h;  depth=d;  }  //constructor  Box()  {  width=-1;  height=-1;  depth=-1;  }  //constructor  Box(double len)  {  width=len;  height=len;  depth=len;  }  double volume()  {  return(width\*height\*depth);  }  }  class Boxweight extends Box  {  double weight;  Boxweight(double w,doubleh,doubled,double m)  {  width=w;  height=h;  depth=d;  weight=m;  }  }  class boxweightdemo  {  public static void main(String args[])  {  Boxweight obj1=new Boxweight(10,20,30,40);  Boxweight obj2=new Boxweight(1,2,3,4);  double vol1,vol2;  vol1=obj1.volume();  vol2=obj2.volume();  System.out.println("volume of box1 is = "+vol1);  System.out.println("weight of box1 is = "+obj1.weight);  System.out.println("volume of box2 is = "+vol2);  System.out.println("weight of box2 is = "+obj2.weight);  }  } |

output

E:\javaprgs>javac boxweightdemo.java

E:\javaprgs>java boxweightdemo

volume of box1 is = 6000.0

weight of box1 is = 40.0

volume of box2 is = 6.0

weight of box2 is = 4.0

**Use of super keyword with variables**

|  |
| --- |
| **class A**  **{**  **String x="welcome";**  **}**  **class B extends A**  **{**  **String x="java";**  **void display()**  **{**  **System.*out.println(x);***  **System.*out.println(super.x);***  **}**  **}**  **public class superkeyword {**  **public static void main(String[] args) {**  **B obj=new B();**  **obj.display();**  **}**  **}** |

Use of super keyword with methods

|  |
| --- |
| **class A**  **{**  **String x="welcome";**  **void display()**  **{**  **System.*out.println(x);***  **}}**  **class B extends A**  **{**  **String x="java";**  **void display()**  **{**  **super.display();**  **System.*out.println(x);***  **}}**  **public class superkeyword {**  **public static void main(String[] args) {**  **B obj=new B();**  **obj.display();**  **}**  **}** |

Use of super keyword with constructor

|  |
| --- |
| **class Account**  **{**  **intaccnum,balance;**  **Account(intaccnum,int balance)**  **{**  **this.accnum=accnum;**  **this.balance=balance;**  **}**  **void credit(int amount)**  **{**  **balance=balance+amount;**  **}**  **void debit(int amount)**  **{**  **balance=balance-amount;**  **}**  **void displaybalance()**  **{**  **System.*out.println("balance = "+balance);***  **}**  **}**  **class SBAccount extends Account**  **{**  **double rate;**  **SBAccount(intaccnum,intbalance,double rate)**  **{**  **super(accnum,balance);**  **this.rate=rate;**  **}**  **void displayrate()**  **{**  **System.*out.println("Interest rate= "+rate);***  **}**  **}**  **public class construct\_inherit {**  **public static void main(String[] args) {**  **SBAccount obj1=new SBAccount(1001,5000,6.5);**  **obj1.credit(2000);**  **obj1.debit(100);**  **obj1.displaybalance();**  **obj1.displayrate();**  **}**  **}** |

|  |
| --- |
| **class** College  {  String collegename,principalname;  College(String cn,Stringpn)  {  **this**.collegename=cn;  **this**.principalname=pn;  }  **void**displaycollegedetails()  {  System.***out***.println("Collegename : "+collegename);  System.***out***.println("Principal name : "+principalname);  }  }  **class** Department **extends** College  {  String deptname,hodname;  Department(String cn,Stringpn, String dept,Stringhod)  {  **super**(cn,pn);  **this**.deptname=dept;  **this**.hodname=hod;  }  **void**displaydeptdetails()  {  System.***out***.println("department : "+deptname);  System.***out***.println("HOD name : "+hodname);  }    }  **publicclass**collegedemo {  **publicstaticvoid** main(String[] args) {  Department obj=**new** Department("SDNB","Varalakshmi","CSC","Sumathi");  obj.displaycollegedetails();  obj.displaydeptdetails();  Department obj1=**new** Department("SDNB","Varalakshmi","Maths","Vijaya");  obj1.displaycollegedetails();  obj1.displaydeptdetails();    }  } |

Another example of using super with constructors

|  |
| --- |
| class Box  {  double width;  double height;  double depth;  //constructor  Box(Box obj)  {  width=obj.width;  height=obj.height;  depth=obj.depth;  }  //constructor  Box(double w, double h,double d)  {  width=w;  height=h;  depth=d;  }  //constructor  Box()  {  width=-1;  height=-1;  depth=-1;  }  //constructor  Box(double len)  {  width=len;  height=len;  depth=len;  }  double volume()  {  return(width\*height\*depth);  }  }  class Boxweight extends Box  {  double weight;  Boxweight(double w,doubleh,doubled,double m)  {  super(w,h,d);  weight=m;  }  }  class boxweightdemo1  {  public static void main(String args[])  {  Boxweight obj1=new Boxweight(10,20,30,40);  Boxweight obj2=new Boxweight(1,2,3,4);  double vol1,vol2;  vol1=obj1.volume();  vol2=obj2.volume();  System.out.println("volume of box1 is = "+vol1);  System.out.println("weight of box1 is = "+obj1.weight);  System.out.println("volume of box2 is = "+vol2);  System.out.println("weight of box2 is = "+obj2.weight);  }  } |

Another example

|  |
| --- |
| class Box  {  double width;  double height;  double depth;  //constructor  Box(Box obj)  {  width=obj.width;  height=obj.height;  depth=obj.depth;  }  //constructor  Box(double w, double h,double d)  {  width=w;  height=h;  depth=d;  }  //constructor  Box()  {  width=-1;  height=-1;  depth=-1;  }  //constructor  Box(double len)  {  width=len;  height=len;  depth=len;  }  double volume()  {  return(width\*height\*depth);  }  }  class Boxweight extends Box  {  double weight;  Boxweight(double w,doubleh,doubled,double m)  {  super(w,h,d);  weight=m;  }  Boxweight(Boxweightobj)  {  super(obj);  weight=obj.weight;  }  Boxweight()  {  super();  weight=-1;  }  Boxweight(double len)  {  super(len);  weight=len;  }  }  class boxweightdemo1  {  public static void main(String args[])  {  Boxweight obj1=new Boxweight(10,20,30,40);  Boxweight obj2=new Boxweight(1,2,3,4);  Boxweight obj3=new Boxweight();  Boxweight obj4=new Boxweight(5.2);  Boxweight obj5=new Boxweight(obj2);  double vol1,vol2,vol3,vol4,vol5;  vol1=obj1.volume();  vol2=obj2.volume();  vol3=obj3.volume();  vol4=obj4.volume();  vol5=obj5.volume();  System.out.println("calling construtor with four variables");  System.out.println("volume of box1 is = "+vol1);  System.out.println("weight of box1 is = "+obj1.weight);  System.out.println("calling construtor with four variables");  System.out.println("volume of box2 is = "+vol2);  System.out.println("weight of box2 is = "+obj2.weight);  System.out.println("calling construtor with -1 value");  System.out.println("volume of box3 is = "+vol3);  System.out.println("weight of box3 is = "+obj3.weight);  System.out.println("calling construtor with one value");  System.out.println("volume of box4 is = "+vol4);  System.out.println("weight of box4 is = "+obj4.weight);  System.out.println("calling construtor with object value");  System.out.println("volume of box5 is = "+vol5);  System.out.println("weight of box5 is = "+obj5.weight);  }  } |

Output

|  |
| --- |
| E:\javaprgs>javac boxweightdemo1.java  E:\javaprgs>java boxweightdemo1  calling construtor with four variables  volume of box1 is = 6000.0  weight of box1 is = 40.0  calling construtor with four variables  volume of box2 is = 6.0  weight of box2 is = 4.0  calling construtor with -1 value  volume of box3 is = -1.0  weight of box3 is = -1.0  calling construtor with one value  volume of box4 is = 140.60800000000003  weight of box4 is = 5.2  calling construtor with object value  volume of box5 is = 6.0  weight of box5 is = 4.0 |

Creating multilevel inheritance

|  |
| --- |
| class Box  {  double width;  double height;  double depth;  //constructor  Box(Box obj)  {  width=obj.width;  height=obj.height;  depth=obj.depth;  }  //constructor  Box(double w, double h,double d)  {  width=w;  height=h;  depth=d;  }  //constructor  Box()  {  width=-1;  height=-1;  depth=-1;  }  //constructor  Box(double len)  {  width=len;  height=len;  depth=len;  }  double volume()  {  return(width\*height\*depth);  }  }  class Boxweight extends Box  {  double weight;  Boxweight(double w,doubleh,doubled,double m)  {  super(w,h,d);  weight=m;  }  Boxweight(Boxweightobj)  {  super(obj);  weight=obj.weight;  }  Boxweight()  {  super();  weight=-1;  }  Boxweight(double len)  {  super(len);  weight=len;  }  }  class shipment extends Boxweight  {  double cost;  shipment(double w,doubleh,doubled,doublem,double c)  {  super(w,h,d,m);  cost=c;  }  shipment(shipment obj)  {  super(obj);  cost=obj.cost;  }  shipment()  {  super();  cost=-1;  }  shipment(double len)  {  super(len);  cost=len;  }  }  class multilevelinherit  {  public static void main(String args[])  {  shipment obj1=new shipment(10,20,30,40,50);  shipment obj2=new shipment(1,2,3,4,5);  shipment obj3=new shipment();  shipment obj4=new shipment(5.2);  shipment obj5=new shipment(obj2);  double vol1,vol2,vol3,vol4,vol5;  vol1=obj1.volume();  vol2=obj2.volume();  vol3=obj3.volume();  vol4=obj4.volume();  vol5=obj5.volume();  System.out.println("calling construtor with four variables");  System.out.println("volume of box1 is = "+vol1);  System.out.println("weight of box1 is = "+obj1.weight);  System.out.println("cost of box1 is = "+obj1.cost);  System.out.println("calling construtor with four variables");  System.out.println("volume of box2 is = "+vol2);  System.out.println("weight of box2 is = "+obj2.weight);  System.out.println("cost of box2 is = "+obj2.cost);  System.out.println("calling construtor with -1 value");  System.out.println("volume of box3 is = "+vol3);  System.out.println("cost of box3 is = "+obj3.cost);  System.out.println("weight of box3 is = "+obj3.weight);  System.out.println("calling construtor with one value");  System.out.println("volume of box4 is = "+vol4);  System.out.println("weight of box4 is = "+obj4.weight);  System.out.println("cost of box4 is = "+obj4.cost);  System.out.println("calling construtor with object value");  System.out.println("volume of box5 is = "+vol5);  System.out.println("weight of box5 is = "+obj5.weight);  System.out.println("cost of box5 is = "+obj5.cost);  }  } |

Output

|  |
| --- |
| E:\javaprgs>javac multilevelinherit.java  E:\javaprgs>java multilevelinherit  calling construtor with four variables  volume of box1 is = 6000.0  weight of box1 is = 40.0  cost of box1 is = 50.0  calling construtor with four variables  volume of box2 is = 6.0  weight of box2 is = 4.0  cost of box2 is = 5.0  calling construtor with -1 value  volume of box3 is = -1.0  cost of box3 is = -1.0  weight of box3 is = -1.0  calling construtor with one value  volume of box4 is = 140.60800000000003  weight of box4 is = 5.2  cost of box4 is = 5.2  calling construtor with object value  volume of box5 is = 6.0  weight of box5 is = 4.0  cost of box5 is = 5.0 |

Another example for multilevel inheritance

|  |
| --- |
| import java.lang.\*;  import java.io.\*;  class Account  {  String cust\_name;  intacc\_no;  Account(String a,int b)  {  cust\_name=a;  acc\_no=b;  }  void display()  {  System.out.println("Customer name: "+cust\_name);  System.out.println("Account no : "+acc\_no);  }  } // end -class Account  class Saving\_Acc extends Account  {  intmin\_bal, saving\_bal;  Saving\_Acc(String a, int b, intc,int d)  {  super(a,b);  min\_bal=c;  saving\_bal=d;  }  void display()  {  super.display();  System.out.println("MInimum Balance: "+min\_bal);  System.out.println("Saving Balance : "+saving\_bal);  }  } // end-class Saving\_Acc  class Acctdetails extends Saving\_Acc  {  int deposits, withdrawals;  Acctdetails(String a, int b, intc,int d, int e, int f)  {  super(a,b,c,d);  deposits=e;  withdrawals=f;  }  void display()  {  super.display();  System.out.println("Deposit: "+ deposits);  System.out.println("withdrawals : "+withdrawals);  }  } //end - class Acctdetails  class multilevel  {  public static void main(String args[])  {  Acctdetailsobj=new Acctdetails("xxx",666,1000,5000,500,9000);  obj.display();  }  } |

Output

|  |
| --- |
| D:\javaprgs>javac multilevel.java  D:\javaprgs>java multilevel  Customer name: xxx  Account no : 666  MInimum Balance: 1000  Saving Balance : 5000  Deposit: 500  withdrawals : 9000 |

**Another example for multilevel inheritance**

|  |
| --- |
| //multilevel inheritance  **import**java.io.\*;  **class**studentdetail  {  **int**rollno;  String stname, brname,year;  studentdetail(**int**r, String sn,Stringbn,Stringy)  {  rollno=r;  stname=sn;  brname=bn;  year=y;  }  **void**display()  {  System.***out***.println(" \n\t\t\t The student details");  System.***out***.println("student rollno : "+rollno);  System.***out***.println("student name : "+stname);  System.***out***.println("student branch : "+brname);  System.***out***.println("student year : "+year);  }  }  **class** mark **extends**studentdetail  {  String category;  **int**mark1,mark2,mark3;  mark(**int**r,Stringsn,Stringbn,Stringy,Stringcat,**int**m1,**int**m2,**int**m3)  {  **super**(r,sn,bn,y);  category=cat;  mark1=m1;  mark2=m2;  mark3=m3;  }  **void**display()  {  **super**.display();  System.***out***.println("student category : "+category);  System.***out***.println("mark1 : "+mark1);  System.***out***.println("mark2 : "+mark2);  System.***out***.println("mark3 : "+mark3);  }  }  **class** report **extends** mark  {  **double**total,avg;  String grade;  report(**int**r,Stringsn,Stringbn,Stringy,Stringcat,**int**m1,**int**m2,**int**m3)  {  **super**(r,sn,bn,y,cat,m1,m2,m3);  }  **void**tot()  {  total=mark1+mark2+mark3;  avg=total/3.0;  **if** ((mark1>=40) && (mark2>=40) && (mark3>=40))  {  **if**(avg>=80)  grade="Distinction";  **elseif**((avg>=60) &&(avg<80))  grade="First class";  **else**  grade="Second class";  }  **else**  grade="Fail";    }  **void**display()  {  tot();  **super**.display();  System.***out***.println("total = "+total);  System.***out***.println("average = "+avg);  System.***out***.println("Grade = "+grade);  }  }  **publicclass** multilevel1 {  **publicstaticvoid** main(String[] args) **throws**IOException  {  **int**id,m1,m2,m3;  String name,br,cat,yr;  **char**ch;  DataInputStreamdin=**new**DataInputStream(System.***in***);  **do**  {  System.***out***.println("Enter student id");  id=Integer.*parseInt*(din.~~readLine~~());  System.***out***.println("Enter student name");  name=din.~~readLine~~();  System.***out***.println("Enter branch name");  br=din.~~readLine~~();  System.***out***.println("Enter category");  cat=din.~~readLine~~();  System.***out***.println("Enter year");  yr=din.~~readLine~~();  System.***out***.println("Enter mark1");  m1=Integer.*parseInt*(din.~~readLine~~());  System.***out***.println("Enter mark2");  m2=Integer.*parseInt*(din.~~readLine~~());  System.***out***.println("Enter mark3");  m3=Integer.*parseInt*(din.~~readLine~~());  report obj=**new** report(id,name,br,yr,cat,m1,m2,m3);  obj.display();  System.***out***.println("want to continue? (Y/N)");  ch=(din.~~readLine~~()).charAt(0);  } **while** (ch=='y' || ch=='Y');  }  } |

**Output**

|  |
| --- |
| Enter student id  100  Enter student name  anitha  Enter branch name  csc  Enter category  day  Enter year  first  Enter mark1  90  Enter mark2  80  Enter mark3  90  The student details  student rollno : 100  student name :anitha  student branch :csc  student year : first  student category : day  mark1 : 90  mark2 : 80  mark3 : 90  total = 260.0  average = 86.66666666666667  Grade = Distinction  want to continue? (Y/N) |

**Constructor order execution**

|  |
| --- |
| **class A**  **{**  **A()**  **{**  **System.out.println("this is A's constructor");**  **}**  **}**  **class B extends A**  **{**  **B()**  **{**  **System.out.println("this is B's constructor");**  **}**  **}**  **class C extends B**  **{**  **C()**  **{**  **System.out.println("this is C's constructor");**  **}**  **}**  **class constructororder**  **{**  **public static void main(String args[])**  **{**  **C obj=new C();**  **}**  **}** |

**output**

|  |
| --- |
| **E:\javaprgs>javac constructororder.java**  **E:\javaprgs>java constructororder**  **this is A's constructor**  **this is B's constructor**  **this is C's constructor** |

**Method overriding in constructor classes**

|  |
| --- |
| **class A**  **{**  **inti,j;**  **A(inta,int b)**  **{**  **i=a;**  **j=b;**  **}**  **void show()**  **{**  **System.out.println("this is from class A ");**  **System.out.println("i = "+i+" j = "+j);**  **}**  **}**  **class B extends A**  **{**  **int k;**  **B(inta,int b, int c)**  **{**  **super(a,b);**  **k=c;**  **}**  **void show()**  **{**  **System.out.println("this is from class B ");**  **System.out.println("k = "+k);**  **}**  **}**  **class methodoverriding**  **{**  **public static void main(String args[])**  **{**  **B obj=new B(1,2,3);**  **obj.show();**  **}**  **}** |

**Output**

|  |
| --- |
| this is from class B  k = 3 |

**To access the base class overriding methods use super**

|  |
| --- |
| **class A**  **{**  **inti,j;**  **A(inta,int b)**  **{**  **i=a;**  **j=b;**  **}**  **void show()**  **{**  **System.out.println("this is from class A ");**  **System.out.println("i = "+i+" j = "+j);**  **}**  **}**  **class B extends A**  **{**  **int k;**  **B(inta,int b, int c)**  **{**  **super(a,b);**  **k=c;**  **}**  **void show()**  **{**  **super.show(); // this calls the parent class method**  **System.out.println("this is from class B ");**  **System.out.println("k = "+k);**  **}**  **}**  **class methodoverriding**  **{**  **public static void main(String args[])**  **{**  **B obj=new B(1,2,3);**  **obj.show();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac methodoverriding.java**  **E:\javaprgs>java methodoverriding**  **this is from class A**  **i = 1 j = 2**  **this is from class B**  **k = 3** |

**Superclass variable can reference a subclass object**

|  |
| --- |
| **//superclass variable referenc a subclass object**  **class A**  **{**  **inti,j;**  **A(inta,int b)**  **{**  **i=a;**  **j=b;**  **}**  **void show()**  **{**  **System.out.println("i = "+i+" j= "+j);**  **}**  **}**  **class B extends A**  **{**  **int k;**  **B(inta,intb,int c)**  **{**  **super(a,b);**  **k=c;**  **}**  **void show()**  **{**  **System.out.println("k = "+k);**  **}**  **}**  **class C extends B**  **{**  **int z;**  **C(inta,intb,intc,int d)**  **{**  **super(a,b,c);**  **z=d;**  **}**  **void show()**  **{**  **System.out.println("z = "+z);**  **}**  **}**  **class refdemo**  **{**  **public static void main(String args[])**  **{**  **B subobj=new B(1,2,3);**  **C subobj2=new C(1,2,3,4);**  **A superobj;**  **subobj.show();**  **superobj=subobj;**  **superobj.show();**  **superobj=subobj2;**  **superobj.show();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac refdemo.java**  **E:\javaprgs>java refdemo**  **k = 3**  **k = 3**  **z = 4** |

**Dynamic method dispatch**

**Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than at compile time and this is how java implements run-time polymorphism.**

**A super class reference variable can refer to a subclass object**

|  |
| --- |
| **class A**  **{**  **void callme()**  **{**  **System.out.println("Inside A");**  **}**  **}**  **class B extends A**  **{**  **void callme() //overriding**  **{**  **System.out.println("Inside B");**  **}**  **}**  **class C extends B**  **{**  **void callme()**  **{**  **System.out.println("Inside C");**  **}**  **}**  **class dispatch**  **{**  **public static void main(String args[])**  **{**  **A a=new A();**  **B b=new B();**  **C c=new C();**  **A r; //obtain a reference of type A**  **r=a; // r referes to an A object**  **r.callme();**  **r=b;**  **r.callme();**  **r=c;**  **r.callme();**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs>java dispatch**  **Inside A**  **Inside B**  **Inside C** |

**Overridden methods achieve run-time polymorphism**

|  |
| --- |
| **// using run-time polymorphism**  **class Figure**  **{**  **double dim1;**  **double dim2;**  **Figure(double a, double b)**  **{**  **dim1=a;**  **dim2=b;**  **}**  **double area()**  **{**  **System.out.println("area of figure");**  **return 0;**  **}**  **}**  **class Rectangle extends Figure**  **{**  **Rectangle(double a, double b)**  **{**  **super(a,b);**  **}**  **//override area in superclass**  **double area()**  **{**  **System.out.println("area of rectangle");**  **return dim1\*dim2;**  **}**  **}**  **class Triangle extends Figure**  **{**  **Triangle(double a,double b)**  **{**  **super(a,b);**  **}**  **double area()**  **{**  **System.out.println("area of triangle");**  **return dim1\*dim2/2;**  **}**  **}**  **class Findarea**  **{**  **public static void main(String args[])**  **{**  **Figure f=new Figure(10,10);**  **Rectangle r=new Rectangle(20,20);**  **Triangle t=new Triangle(30,30);**  **Figure figref;**  **figref=r;**  **System.out.println("Area of rectangle "+figref.area());**  **figref=t;**  **System.out.println("Area of triangle "+figref.area());**  **figref=f;**  **System.out.println("Area of figure "+figref.area());**  **}**  **}** |

**Output**

|  |
| --- |
| **E:\javaprgs>javac Findarea.java**  **E:\javaprgs>java Findarea**  **area of rectangle**  **Area of rectangle 400.0**  **area of triangle**  **Area of triangle 450.0**  **area of figure**  **Area of figure 0.0** |

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

**A class for which we cannot create objects is called abstract class. An abstract class is to be extended and then instantiated.**

Another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. You don't know the internal processing about the message delivery.

Example abstract class

1. **abstract** **class** A{}

abstract method

|  |
| --- |
| A method that is declared as abstract and does not have implementation is known as abstract method. |

Example abstract method

**abstract** **void** printStatus();//no body and abstract

The following will give error.

|  |
| --- |
| **abstractclass** A  {    }  **publicclass**abstractdemo {  **publicstaticvoid** main(String[] args) {  A obj=**new**obj();  }  } |

Exception in thread "main" java.lang.Error: Unresolved compilation problem:

Cannot instantiate the type A

at Construct\_inherit.abstractdemo.main(abstractdemo.java:10)

|  |
| --- |
| **abstractclass** A  {    }  **class** B **extends** A  {  **Void** display()  {  System.***out***.println("welcome");  }  }  **public class** abstractdemo {  **public static void** main(String[] args) {  B obj=**new** B();  obj.display();  }  } |

Use of abstract method

|  |
| --- |
| **abstract class** A1  {  a**bstract void** show1();  }  **Class** B1 **extends** A1  {    **void** show1()//abstract method-coding is given here  {  System.***out***.println("this is abstract method");  }  }  **publicclass** abstractdemo2 {  **public static void** main(String[] args) {  B1obj=**new**B1();  obj.show1();  }  } |

Abstract class, abstract method with ordinary method

|  |
| --- |
| **abstractclass** A1  {  **Abstract void** show1();  **void** show2()  {  System.***out***.println("welcome");  }  }  **class** B1 **extends** A1  {    **void** show1()  {  System.***out***.println("this is abstract method");  }  }  **publicclass** abstractdemo3 {  **publicstaticvoid** main(String[] args) {  B1 obj=**new** B1();  obj.show1();  obj.show2();  }  } |

An abstract method should be defined inside an abstract class only. For eg

class A

{

abstract void show();

}

The above declaration will produce an error.

Example :Abstract class inherited in more than one class

|  |
| --- |
| **abstract clas s**compcentre  {  a**bstract int** numcomp();  }  **class** centre1 **extends** compcentre  {  **int** numcomp()  {  **return** 45;  }  }  **class** centre2 **extends** compcentre  {  i**nt** numcomp()  {  **return** 11;  }  }  **publicclass** abstractdemo4 {  **publicstaticvoid** main(String[] args) {  centre1 obj1=**new** centre1();  centre2 obj2=**new** centre2();  System.***out***.println("the number of computers in centre1 is= "+obj1.numcomp());  System.***out***.println("the number of computers in centre2 is= "+obj2.numcomp());  }  } |

Another example

|  |
| --- |
| **abstractclass** Shape  {  **abstractvoid**draw();  }  **class** Rectangle **extends** Shape  {  **void**draw()  {  System.***out***.println("drawing rectangle");  }  }  **class** circle **extends** Shape  {  **void**draw()  {  System.***out***.println("drawing circle");  }  }  **publicclass** abstractdemo6 {  **publicstaticvoid** main(String[] args) {  circle s=**new**circle();  Rectangle r=**new**Rectangle();  s.draw();  r.draw();  }  } |

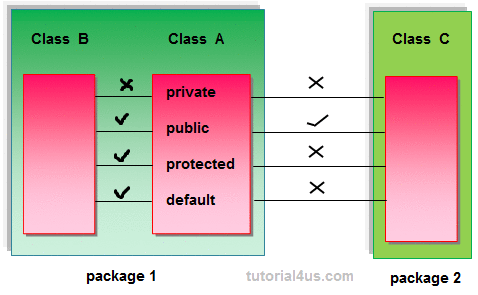
Abstract class with constructor, ordinary method, abstract method

|  |
| --- |
| **abstractclass** Bike  {  Bike()  {  System.***out***.println("bike is created");  }  **abstractvoid**run();  **void**changegear()  {  System.***out***.println("gear changed");  }    }  **class**Honda**extends** Bike  {  **void**run()  {  System.***out***.println("bike is running");  }  }  **publicclass** abstractdemo5 {  **publicstaticvoid** main(String[] args) {  Hondaobj=**new**Honda();  obj.run();  obj.changegear();  }  } |

Access specifiers / modifiers

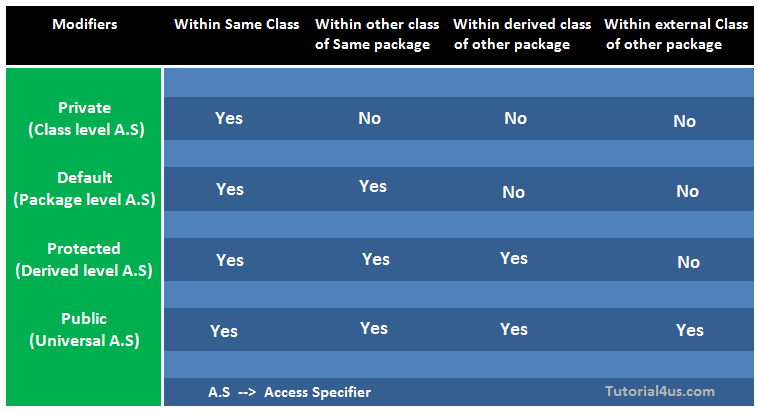
**Access modifiers**are those which are applied before data members or methods of a class. These are used to where to access and where not to access the data members or methods. In Java programming these are classified into four types:

* Private
* Default (not a keyword)
* Protected
* Public



Protected members of the class are accessible within the same class and another class of same package and also accessible in inherited class of another package.

## Rules for access modifiers:



**private:**Private members of class in not accessible anywhere in program these are only accessible within the class. Private are also called class level access modifiers

|  |
| --- |
| // program for private access  class hello  {  private int a=10;  private void show()  {  System.out.println("hello java");  }  }  public class privatedemo  {  public static void main(String args[])  {  hello obj=new hello();  System.out.println(obj.a);  obj.show();  }  } |

**Output**

E:\javaprgs>javac privatedemo.java

privatedemo.java:16: a has private access in hello

System.out.println(obj.a);

^

privatedemo.java:17: show() has private access in hello

obj.show();

^

2 errors.

**public:**Public members of any class are accessible anywhere in the program in the same class and outside of class, within the same package and outside of the package. Public are also called universal access modifiers.

|  |
| --- |
| // program for public access  class hello  {  public int a=10;  public void show()  {  System.out.println("hello java");  }  }  public class publicdemo  {  public static void main(String args[])  {  hello obj=new hello();  System.out.println(obj.a);  obj.show();  }  } |

**output**

E:\javaprgs>javac publicdemo.java

E:\javaprgs>java publicdemo

10

hello java

Program which use both public and private access

|  |
| --- |
| // program for public-private access  class hello  {  public int a=10;  public void show()  {  System.out.println("hello java");  }  private void testshow()  {  System.out.println("this function is private and cannt be accessed");  }  }  public class publicprivatedemo  {  public static void main(String args[])  {  hello obj=new hello();  System.out.println(obj.a);  obj.show();  obj.testshow();  }  } |

**output**

E:\javaprgs>javac publicprivatedemo.java

publicprivatedemo.java:23: testshow() has private access in hello

obj.testshow();

^

1 error

Use of protected mode

|  |
| --- |
| // program for protected access  class hello  {  protected int a=10;  public void show()  {  System.out.println("hello java");  }  protected void testshow()  {  System.out.println("now this function is protected and can be accessed through derived class");  }  }  class childhello extends hello  {  }  public class protecteddemo  {  public static void main(String args[])  {  hello obj=new hello();  System.out.println(obj.a);  obj.show();  obj.testshow();  }  } |

Output

E:\javaprgs>javac protecteddemo.java

E:\javaprgs>java protecteddemo

10

hello java

now this function is protected and can be accessed through derived class

final keyword in Java

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

**Final variable**

A variable declared as final cannot be changed. (constant)

The final keyword

1. Stop value change
2. Stop method overriding
3. Stop inheritance

In this example the variable pi is declared as final but it is again assigned some other value in the function area. This gives the error

|  |
| --- |
| class circle1  {  final double pi=3.14;  double area(int r)  {  pi=45;  return(pi\*r\*r);  }  }  class finaleg1  {  public static void main(String args[])  {  circle1 obj=new circle1();  double a;  a=obj.area(5);  System.out.println("area is = "+a);  }  } |

Output

|  |
| --- |
| D:\javaprgs>javac finaleg1.java  finaleg1.java:6: error: cannot assign a value to final variable pi  pi=45;  ^ |

**Java final method**

A method which is defined as final cannot be overridden

|  |
| --- |
| class circle1  {  final double pi=3.14;  final double area(int r)  {  return(pi\*r\*r);  }  }  class rectangle extends circle1  {  double area(int l)  {  return(l\*l);  }  }  class finaleg2  {  public static void main(String args[])  {  rectangle obj=new rectangle();  double a;  a=obj.area(5);  System.out.println("area is = "+a);  }  } |

Output

|  |
| --- |
| D:\javaprgs>javac finaleg2.java  finaleg2.java:12: error: area(int) in rectangle cannot override area(int) in cir  cle1  double area(int l)  ^  overridden method is final  1 error |

**Java final class**

A class that is declared with the final modifier cannot be **extended**. This means that such a class cannot have any subclasses. The following program will generate an error

|  |
| --- |
| final class shape2d  {  void display()  {  System.out.println("this is shape2d final class");  }  }  class triangle extends shape2d  {  void display()  {  System.out.println("this is triangle class");  }  }  class finaleg3  {  public static void main(String args[])  {  triangle obj=new triangle();  obj.display();  }  } |

Output

|  |
| --- |
| D:\javaprgs>javac finaleg3.java  finaleg3.java:9: error: cannot inherit from final shape2d  class triangle extends shape2d  ^  1 error |