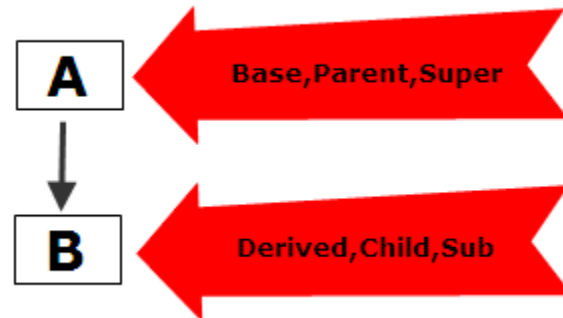


Inheritance

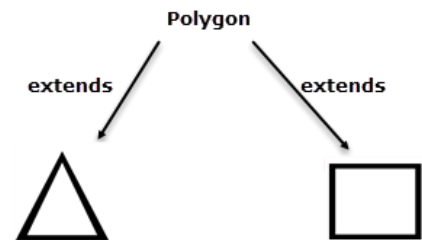
- Inheritance means heredity. Some of the characteristics of son or daughter are common to their parents. Inheritance can be defined as the process where one class acquires the methods and fields of another.
- Inheritance means creating new classes based on an existing class. The new class will have combined features of both the classes.
- The class which inherits the properties of other is known as subclass or derived class or child class and the class whose properties are inherited is known as superclass or base class or parent class. In above image, A is Super class and B is Derived class of A.
- A Derived class receives all the methods and properties of the base class. The size of the sub class is always bigger than the size of super class's object.
- It provides reusability.
- It is also known as generalization.
- Inheritance represents "is-a" relationship. Abhishek Bachchan is a son of Amitabh Bachchan.
- All classes are inheritable by default.
- For example, Triangle and Rectangle share the characteristics of Polygon (Height, Width field and Area() method). But the formula of Area make them different: formula of Area of Triangle is $(\text{Height} * \text{Width} * 0.5)$ and Area of Rectangle is $(\text{Height} * \text{Width})$.
- Object-oriented programming allows classes to inherit field and method from an existing class.
- "extends" keyword** is used to create relationship between classes. Extends is the keyword used to inherit the properties of a class.



```
class poly
{
}

class Triangle extends Poly{
    // new data members and data functions defining
}

class Rectangle extends Poly{
    // new data members and data functions defining
}
```



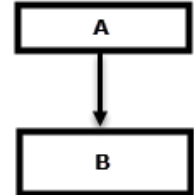
Advantages of Inheritance

- ✦ Reusability.
- ✦ Saves time, in terms of verification and testing.
- ✦ Aids in modularization of code.
- ✦ Better organization of code.

Types of Inheritance

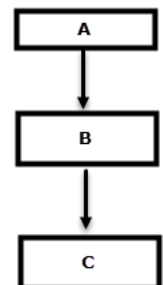
Single Inheritance

One base class and one derived class. For example, Base class A is inherited by class B.



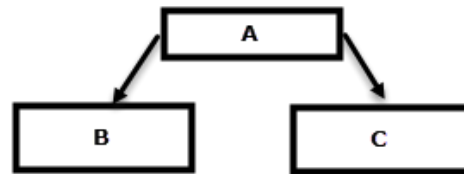
Multi-level Inheritance

Chain of inheritance. For example Base class A is inherited by Derived class B, B is inherited by another Derived class C.



Hierarchical Inheritance

One base class and two derived class. For example class A is inherited by B and C both



Example of Single Inheritance

Class Base contains data member a and two methods setA() and printA() then derived class Derived contains a and three methods setA() , printB() and Multi(). Create an object of derived class and call all above methods.

Example# Demo of Single Inheritance

Output View

```
E:\javaDemo>java jInh1
Enter no A =>5
Enter no B =>10
A = 5
B = 10
Multi = 50
```

Coding View

```
import java.util.*;
```

```
class Base
```

```
{
    int a;
    void setA()
    {
        Scanner s1=new Scanner(System.in);

        System.out.print("Enter no A =>");
        a=s1.nextInt();
    }
    void printA()
    {
        System.out.println("A = " +a);
    }
}
```

```
class Derived extends Base
```

```
{
    int b;
    void setB()
```



```
{
    Scanner s1=new Scanner(System.in);

    System.out.print("Enter no B =>");
    b=s1.nextInt();
}
void printB()
{
    System.out.println("B = " +b);
}
void multi()
{
    System.out.println("Multi = " + (a*b));
}
}
```

class jInh1

```
{
    public static void main(String args[])
    {
        Derived d1=new Derived();
        d1.setA();
        d1.setB();
        d1.printA();
        d1.printB();
        d1.multi();
    }
}
```

Example of Multilevel Inheritance

Class Student contains two data members sno, sname and two data functions setStu(), printStu(). Marks is a derived class of Student which contains two data members Hindi, English and two data functions setMarks() , printMarks(). Result is a derived class of Marks which contains one data member total and data function printTotal(). Create an object of Result class and call all the methods.

Example# Demo of Multilevel Inheritance**Output View**



```
E:\javaDemo>javac jInh2.java

E:\javaDemo>java jInh2
Enter sno =>11
Enter sname =>Ram
Enter eng =>22
Enter hindi =>33
Sno = 11 Sname = Ram
English = 22 Hindi = 33
Total = 55
```

Coding View

```
import java.util.*;
```

class Stu

```
{
    int sno;
    String sname;

    void setStu()
    {
        Scanner s1=new Scanner(System.in);
        System.out.print("Enter sno =>");
        sno=s1.nextInt();

        System.out.print("Enter sname =>");
        sname=s1.next();
    }
    void printStu()
    {
        System.out.println("Sno = " + sno + " Sname = " + sname);
    }
}
```

class Marks extends Stu

```
{
    int eng,hindi;

    void setMarks()
    {
        Scanner s1=new Scanner(System.in);
        System.out.print("Enter eng =>");
        eng=s1.nextInt();
        System.out.print("Enter hindi =>");
        hindi=s1.nextInt();
    }
    void printMarks()
```





```
    {  
        System.out.println("English = " + eng+" Hindi = " + hindi);  
    }  
}
```

class Result extends Marks

```
{  
    int total;  
    void printTotal()  
    {  
        total=eng+hindi;  
        System.out.println("Total = " + total);  
    }  
}
```

class jInh2

```
{  
    public static void main(String args[])  
    {  
        Result r1=new Result();  
        r1.setStu();  
        r1.setMarks();  
        r1.printStu();  
        r1.printMarks();  
        r1.printTotal();  
    }  
}
```

Example of Hierarchical Inheritance

Create a class Poly which contains two data members height, width and two data functions setData() , printData(). Create two derived class triangle and rectangle which contains area() methods respectively. Now create an object of Triangle and Rectangle and call the above functions.

Example# Demo of Hierarchical Inheritance**Output View**



```
E:\javaDemo>javac jInh3.java

E:\javaDemo>java jInh3
Enter Height =>100
Enter Width =>200
Height = 100.0 Width = 200.0
Area of Rect = 20000.0
Enter Height =>100
Enter Width =>200
Height = 100.0 Width = 200.0
Area of Tri = 10000.0
```

Coding View

```
import java.util.*;
```

class Poly

```
{
    double h,w;
    void setData()
    {
        Scanner s1=new Scanner(System.in);
        System.out.print("Enter Height =>");
        h=s1.nextDouble();

        System.out.print("Enter Width =>");
        w=s1.nextDouble();
    }
    void printData()
    {
        System.out.println("Height = " + h+ " Width = " + w);
    }
}
```

class Tri extends Poly

```
{
    void printArea()
    {
        System.out.println("Area of Tri = " + (h*w*0.5));
    }
}
```

class Rect extends Poly

```
{
    void printArea()
    {
        System.out.println("Area of Rect = " + (h*w));
    }
}
```



class jInh3

```
{  
    public static void main(String args[])  
    {  
        Rect r1=new Rect();  
        r1.setData();  
        r1.printData();  
        r1.printArea();  
  
        Tri t1=new Tri();  
        t1.setData();  
        t1.printData();  
        t1.printArea();  
    }  
}
```

Constructor in Inheritance

It is the responsibility of derived class to call base class's constructor. It means base class constructors are always called in the derived class constructors.

Whenever we create derived class object, first the base class constructor is executed and then the derived class's constructor.

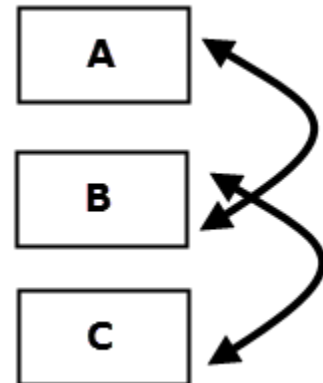
A derived class's constructor's first task is to call its superclass' constructor.

There could be any number of base classes in an inheritance chain. We can say it as constructor chaining.

As we know we can declare two types of constructor - with argument and without argument.

We can call constructor from the derived class by class with super and without super keyword.

If a Base class is having without argument constructor then it gets automatically called in Derived class constructor.

**Example# Demo of without argument constructor in Inheritance****Output View**



```
E:\javaDemo>javac jInhConstructor.java

E:\javaDemo>java jInhConstructor
Hi in A Class Constructor
Hi in B Class Constructor
Hi in C Class Constructor
```

Coding View

```
class A{
    A() {
        System.out.println("Hi in A Class Constructor");
    }
}

class B extends A {
    B() {
        System.out.println("Hi in B Class Constructor");
    }
}

class C extends B {
    C() {
        System.out.println("Hi in C Class Constructor");
    }
}

public class jInhConstructor {
    public static void main(String[] args) {
        C c1=new C();
    }
}
```

If Base class is having with argument constructor then we need to take help of super keyword to call from Derived class constructor. Call of parameterized constructor must have Super(arguments...) in the first line in the Derived class constructor.

Example# Demo of with argument constructor in Inheritance

Output View

```
E:\javaDemo>javac jInhConstructor2.java

E:\javaDemo>java jInhConstructor2
Hi in A Class Constructor
Hi in B Class Constructor
Hi in C Class Constructor
```

Coding View





```
class A{

int x,y;

A(int a,int b) {
    x=a;
    y=b;
    System.out.println("Hi in A Class Constructor");
}
}

class B extends A {
    int p,q;

B(int a,int b,int c,int d) {
    super(a,b);
    p=c;
    q=d;
    System.out.println("Hi in B Class Constructor");
}
}

class C extends B {
    int r;

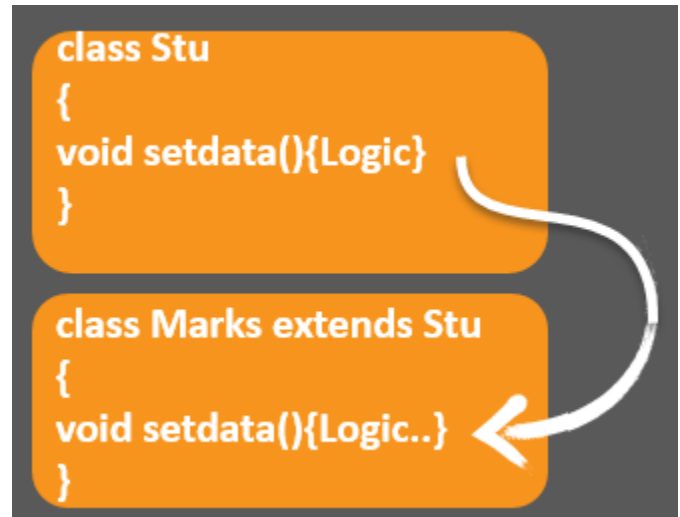
C(int a,int b,int c,int d,int e) {
    super(a,b,c,d);
    r=e;
    System.out.println("Hi in C Class Constructor");
}
}

public class jInhConstructor2 {
    public static void main(String[] args) {
        C c1=new C(11,22,33,44,55);
    }
}
```



Method Overriding

- Method Overriding is used when we want method of same name in both base and derived class.
- When we declare a method in subclass which is already present in base class is known as method overriding.
- Method overriding is the example of polymorphism in OOPS which allows programmer to create two methods with same name and method signature in a derived class.
- Overriding method has its own specific implementation to an inherited method without even modifying the base class.
- In Method overloading, only name of two overloaded methods are same but method signature means number of arguments or sequence of arguments or data types must be different while in Method overriding, method all things are same like argument list, data type, access modifier etc.
- Private and final methods cannot be overridden.
- If a class is extending an abstract class or implementing an interface then it has to (compulsory) override all the abstract methods.



Example# Demo of Method Overriding in Inheritance

Output View

```
E:\javaDemo>javac MethodOverDemo.java

E:\javaDemo>java Demo
Enter height =>100
Enter width =>200
Area of Triangle = 10000.0
```

Coding View

```
import java.util.*;

class Polygon
{
    double h,w;
```



```

void setdata()
{
    Scanner s1=new Scanner(System.in);
    System.out.print("Enter height =>");
    h=s1.nextDouble();
    System.out.print("Enter width =>");
    w=s1.nextDouble();
}

void area()
{
    System.out.println("\nArea");
}

class Tri extends Polygon
{
    void area()
    {
        System.out.println("Area of Triangle = " + (h*w*0.5));
    }
}

class Demo
{
    public static void main(String args[])
    {
        Tri t1=new Tri();
        t1.setdata();
        t1.area();
    }
}

```

In above program, we override area() method. Will see in depth with super keyword.

| |
|--|
| Method Overloading Vs Method Overriding |
|--|

| Method Overloading | Method Overriding |
|---|--|
| Method overloading is performed within same class. | Method overriding occurs in two classes with inheritance relationship. |
| Example of compile time polymorphism. | Example of run time polymorphism. |
| Parameters must be different. | Parameters are same. |
| It does not hide any method. | It hides base class method. |
| The overloaded functions may have different return types. | In method overriding all the methods will have the same return types. |
| Private, static and final methods can be overloaded. | Private, static and final methods cannot be override. |
| Example# | Example# |





```
class Math{  
void add(int a,int b){Logic...}  
void add(int a,int b,int c){Logic.}  
}
```

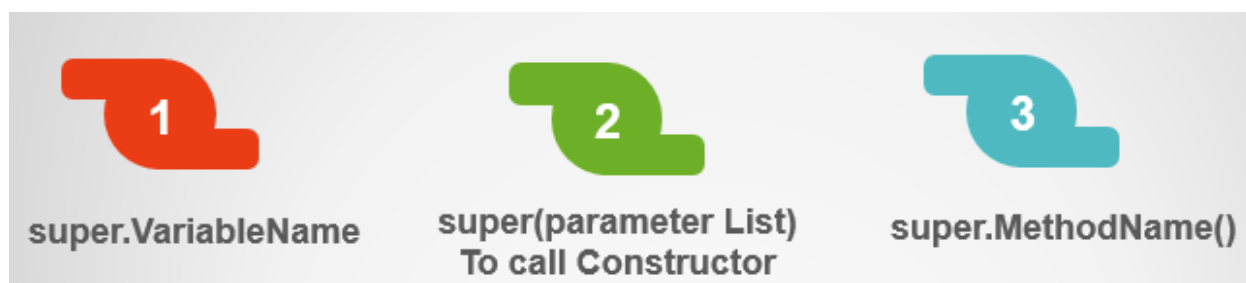
```
class Fish{  
void eat(){Logic...}}  
class JellyFish extends Fish{  
void eat(){Logic...}  
}
```

Super

As the name suggest super keyword is used to access the members of the super class. (Usually I tell students super means upar).

Super is a reference variable that is used to refer parent class object.

There are three situations where we need to use Super.



- 1) `super.VariableName` refers to the variable of parent class to avoid name collision.
- 2) `super(parameterlist)` invokes the constructor of immediate parent class.
- 3) `super.MethodName()` refers to the method of parent class when we use Method Overriding.

`super.VariableName`

If the name of the base class data members are similar to derived class data members and we want to access base class data member in derived class method then we need to write `super` keyword before the base class's data member name.

So to avoid name collision issue, we can call base class's data member using `super.VariableName`

Example# Demo of Super keyword with variable in Inheritance

Output View





```
E:\javaDemo>javac Super1.java
```

```
E:\javaDemo>java Super1
```

```
A = 20
```

```
A = 30
```

```
Multi = 600
```

Coding View

```
import java.util.*;
```

class Fy

```
{
    int a=20;

    void printFy()
    {
        System.out.println("A = "+ a );
    }
}
```

class Sy extends Fy

```
{
    int a=30;

    void printSy()
    {
        System.out.println("A = "+ a );
    }

    void multi()
    {
        System.out.println("Multi = "+ a*super.a );
    }
}
```

class Super1

```
{
    public static void main(String args[])
    {
        Sy s1=new Sy();
        s1.printFy();
        s1.printSy();
        s1.multi();
    }
}
```

In above program, derived class Sy inherits base class Fy's data members and data function and both contain data member 'a'. In order to differentiate between the base class Fy's data



member 'a' and derived class Sy's data member 'a' we need to write Super.a in the derived class method Multi(), if we are not writing super keyword before the base class data member 'a' than it will be referred as current class Sy's data member 'a'.

`super.constructor(parameterList)`

When we create without argument constructor in base and derived class, compiler automatically calls base class's constructor from the derived class's constructor.

Super(parameterList) is used for calling super class parameterized constructor from the derived class's constructor.

Constructor are called from bottom to top but they are executed from top to bottom.

For example – If there are three class A, B and C. class B inherits class A, class C inherits class B. When we create an object of class C (i.e. the last derived class) and it is called, then first the constructor of A is called then the constructor of B is called, then lastly the constructor C is called.

Example# Demo of Super keyword with constructor in Inheritance

Output View

```
E:\javaDemo>javac Super2.java

E:\javaDemo>java Super2
Sno = 1 Sname = Ram
English = 22 Hindi = 33
Total = 55
```

Coding View

```
import java.util.*;

class Stu
{
    int sno;
    String sname;

    Stu(int a,String b)
    {
        sno=a;
        sname=b;
    }

    void printStu()
    {
        System.out.println("Sno = " + sno + " Sname = " + sname);
    }
}

class Marks extends Stu
```



```
{
    int eng,hindi;

    Marks(int a, String b,int x,int y)
    {
        super(a,b);
        eng=x;
        hindi=y;
    }
    void printMarks()
    {
        System.out.println("English = " + eng+" Hindi = " + hindi);
    }
}

class Result extends Marks
{
    int total;

    Result(int a, String b,int x,int y)
    {
        super(a,b,x,y);
        total=x+y;
    }

    void printTotal()
    {
        System.out.println("Total = " + total);
    }
}

class Super2
{
    public static void main(String args[])
    {
        Result r1=new Result(1,"Ram",22,33);
        r1.printStu();
        r1.printMarks();
        r1.printTotal();
    }
}
```

In above program, Stu class is inherited by Marks class and then Marks class is inherited by Result. Student class has constructor with two arguments, Derived class Marks has constructor with four arguments (base + derived) then Derived class of Marks – Result has four arguments (base + derived). Then we create an object of Result and pass values to the base class.

| |
|------------------|
| super.methodName |
|------------------|





If a derived class method overrides one of its superclass's methods, we can call and execute the overridden method through the use of the `super.methodName` from the derived class.

Example# Demo of Super keyword with Method Overriding in Inheritance

Output View

```
E:\javaDemo>javac SuperWithMethod.java

E:\javaDemo>java SuperWithMethod
Enter sno =>12
Enter sname =>Vedika
Enter eng =>22
Enter hindi =>33
Sno = 12 Sname = Vedika
English = 22 Hindi = 33
Total = 55
```

Coding View

```
import java.util.*;

class Stu
{
    int sno;
    String sname;

    void setData()
    {
        Scanner s1=new Scanner(System.in);
        System.out.print("Enter sno =>");
        sno=s1.nextInt();

        System.out.print("Enter sname =>");
        sname=s1.next();
    }

    void printData()
    {
        System.out.println("Sno = " + sno + " Sname = " + sname);
    }
}

class Marks extends Stu
{
    int eng,hindi;

    void setData()
    {
        super.setData();
        Scanner s1=new Scanner(System.in);
```





```
        System.out.print("Enter eng =>");
        eng=s1.nextInt();
        System.out.print("Enter hindi =>");
        hindi=s1.nextInt();
    }
    void printData()
    {
        super.printData();
        System.out.println("English = " + eng+" Hindi = " + hindi);
    }
}

class Result extends Marks
{
    int total;

    void printData()
    {
        super.printData();
        total=eng+hindi;
        System.out.println("Total = " + total);
    }
}

class SuperWithMethod
{
    public static void main(String args[])
    {
        Result r1=new Result();
        r1.setData();
        r1.printData();
    }
}
```

In above program, all the class Student-Marks-Result have same signature of the functions that means this program is an example of function overriding and then we call base class's method using super.methodName() respectively.

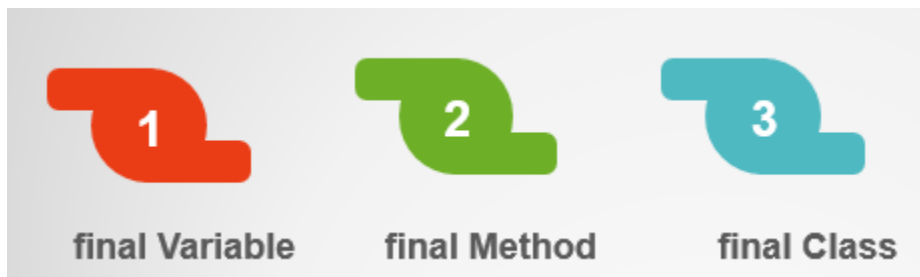
Final

Final means constant.

As the name suggests final keyword is used to make the class, data members and data functions constant or final.

There are three situations where we need to use final.





- 1) final variable is initialized only once.
- 2) final method is restricted to override method.
- 3) final class cannot be inherited.

Final Variable

Final variable is used to make a variable as constant.

Syntax#

```
class className
{
    final datatype dataMemberX=value;
    final datatype dataMemberY;

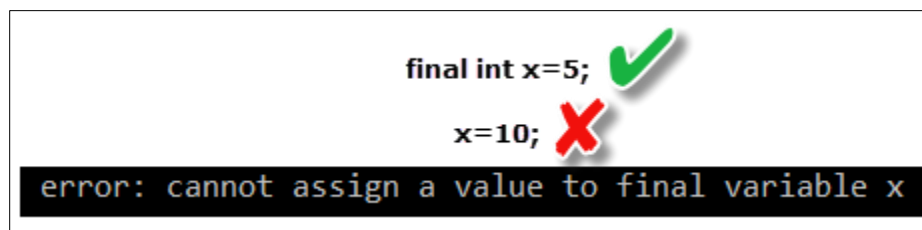
    className()//constructor
    {
        dataMemberY=value;
    }
}
```

Example#

```
class Math
{
    final double PI=3.14;
    final int x;

    Math() //constructor
    {
        X=22;
    }
}
```

The final variable can be assigned only once.



We can initialize final variable either in the constructor or at the time of declaration.

The final variable can be assigned only once. Final variable cannot be modified by the program after initialization.

The compiler checks and gives an error if we try to initialize the final variable once again.



Example# Demo of final keyword with variable in Inheritance

Output View

```
E:\javaDemo>javac finalVar.java

E:\javaDemo>java finalVar
Area of Circle = 31400.0
```

Coding View

```
class Circle
{
    double r;
    final double PI=3.14;

    Circle(double x)
    {
        //Or initialize PI=3.14 here
        r=x;
    }

    void findArea()
    {
        System.out.println("Area of Circle = "+ (PI*r*r));
    }
}

class finalVar
{
    public static void main(String args[])
    {
        Circle r1=new Circle(100);
        r1.findArea();
    }
}
```

Final Method

- 📄 Methods declared as final cannot be overridden.
- 📄 When we want a method which should not be changed and not be re-declared in the derived class that time we can declare method as a final.
- 📄 Final method cannot be overridden in the derived class.

Syntax#

```
class className
{
```

Example#

```
class Math
{
```

```
final returnType methodName()  
{  
    Logic...  
}  
  
}
```

```
final void setData()  
{  
    Logic...  
}  
  
}
```

📄 The compiler checks and gives an error if we try to override the method.

```
class Stu  
{  
    int sno;  
    String sname;  
  
    final void setData()  
    {  
        Logic...  
    }  
    final void printData()  
    {  
        Logic...  
    }  
}  
class Marks extends Stu  
{  
    int eng,hindi;  
  
    void setData()  
    {  
        Logic...  
    }  
    void printData()  
    {  
        Logic...  
    }  
}
```

```
E:\javaDemo>javac finalMethod.java  
finalMethod.java:26: error: setData() in Marks cannot override setData() in Stu  
    void setData()  
        ^  
    overridden method is final  
finalMethod.java:34: error: printData() in Marks cannot override printData() in Stu  
    void printData()  
        ^  
    overridden method is final
```

Example# Demo of final keyword with Method Overriding in Inheritance

Output View



```
E:\javaDemo>javac finalMethod.java

E:\javaDemo>java finalMethod
Enter sno =>12
Enter sname =>ram
Enter eng =>22
Enter hindi =>33
Sno = 12 Sname = ram
English = 22 Hindi = 33 Total = 55
```

Coding View

```
import java.util.*;
```

```
class Stu
{
```

```
    int sno;
    String sname;
```

final void setData()

```
{
    Scanner s1=new Scanner(System.in);
    System.out.print("Enter sno =>");
    sno=s1.nextInt();

    System.out.print("Enter sname =>");
    sname=s1.next();
}
```

final void printData()

```
{
    System.out.println("Sno = " + sno + " Sname = " + sname);
}
```

```
}
class Marks extends Stu
{
```

```
    int eng,hindi;
```

void setMarks()

```
{
    Scanner s1=new Scanner(System.in);
    System.out.print("Enter eng =>");
    eng=s1.nextInt();
    System.out.print("Enter hindi =>");
    hindi=s1.nextInt();
}
```

void printMarks()

```
{
    System.out.println("English = " + eng+" Hindi = " + hindi + " Total = " +
(eng+hindi));
}
```





```
    }  
}  
  
class finalMethod  
{  
    public static void main(String args[])  
    {  
        Marks m1=new Marks();  
        m1.setData();  
        m1.setMarks();  
        m1.printData();  
        m1.printMarks();  
    }  
}
```

Final Class


 Final classes cannot be extended.

Syntax#

```
final class className  
{  
    returntype methodName()  
    {  
        Logic...  
    }  
}
```

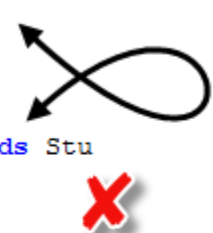
Example#

```
final class Math  
{  
    void setData()  
    {  
        Logic...  
    }  
}
```

 When we want to secure a class i.e. cannot be inherited by any class or say restrict inheritance, then we make that class as final.



```
final class Stu
{
    //Logic....
}
class Marks extends Stu
{
    //Logic...
}
```



```
finalMethodWrong.java:22: error: cannot inherit from final Stu
class Marks extends Stu
      ^
1 error
```

Example# Demo of final class in Inheritance

Output View

```
E:\javaDemo>javac finalClass.java

E:\javaDemo>java finalClass
Enter sno =>1
Enter sname =>Ravi
Sno = 1 Sname = Ravi
```

Coding View

```
import java.util.*;
```

final class Stu

```
{
    int sno;
    String sname;
    void setData()
    {
        Scanner s1=new Scanner(System.in);
        System.out.print("Enter sno =>");
        sno=s1.nextInt();

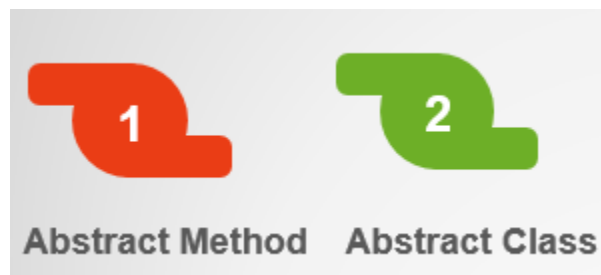
        System.out.print("Enter sname =>");
        sname=s1.next();
    }
    void printData()
    {
        System.out.println("Sno = " + sno + " Sname = " + sname);
    }
}
```



```
class finalClass
{
    public static void main(String args[])
    {
        Stu s1=new Stu();
        s1.setData();
        s1.printData();
    }
}
```


Abstract Class and Abstract Method

📄 We can use abstract keyword with



Abstract Method

- 📄 An abstract method consists of a method signature, but does not have a method body.
- 📄 An abstract methods ends with ";" instead of logic between curly braces.

 **abstract void add(int a,int b);**
- 📄 Abstract methods are used to provide a template for the classes that inherit the abstract methods.
- 📄 Abstract method should be compulsorily redefined in the Derived class.
- 📄 Abstract methods must be overridden.
- 📄 All sub classes of abstract class, should either override the abstract method or declare itself as an abstract class.
- 📄 An abstract method cannot be contained in a non-abstract class.

Abstract Class

📄 A class which contains at least one abstract method should be declared as an abstract class.

📄 An abstract class may or may not have abstract methods. Means it is possible to define an abstract class that doesn't contain any abstract methods. But usually we have abstract class with abstract methods.



```
abstract class force{  
    abstract void add(int a,int b);  
    abstract void mul(int a,int b);  
    void setdata(){Logic...}  
}
```

📄 An abstract class is one that cannot be instantiated. An abstract class has no use until it is extended by some other class. Abstract method should compulsorily be redefined in the Derived class.

📄 An abstract class contains data members, non-abstract data functions, constructors etc.

📄 Constructors of an abstract class are invoked by its subclasses.

📄 Abstract classes are used to declare common characteristics of subclasses.

Syntax#

```
abstract className  
{  
    datatype dataMember;  
    abstract returnType functionName();  
    returnType functionName() {Logic..}  
}
```

Example#

```
abstract classForce  
{  
    int x,y;  
    abstract void area(int a,int b);  
    void setdata(){Logic...}  
}
```

Example# Demo of abstract class and abstract method

Output View

```
E:\javaDemo>javac Abstract1.java  
  
E:\javaDemo>java Abstract1  
Tri = 10000.0  
Rect = 20000
```

Coding View

abstract class shape

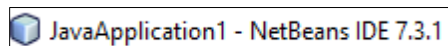
```
{  
    abstract public void area(int h,int w);  
}
```

```
class Tri extends shape
{
    public void area(int h,int w)
    {
        System.out.println("Tri = " + (0.5*h*w));
    }
}

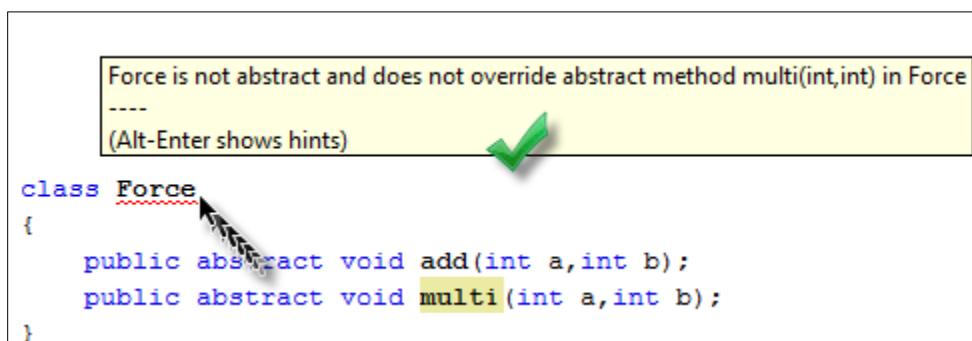
class Rect extends shape
{
    public void area(int h,int w)
    {
        System.out.println("Rect = " + (h*w));
    }
}

class Abstract1
{
    public static void main(String args[])
    {
        Tri t1=new Tri();
        Rect r1=new Rect();
        t1.area(100,200);
        r1.area(100,200);
    }
}
```

Let see the practical Demo of Abstract class and Abstract method, step by step in Smart Editor like NetBeans



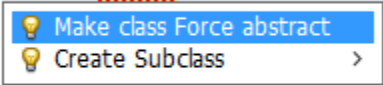
Step1: Create a class Force which extends without Abstract class keyword, so it shows an error.



Step 2: Now click on Force Class, it shows options.



```
14 class Force
15
16     add(int a,int b);
17     public abstract void multi(int a,int b);
18 }
19
```

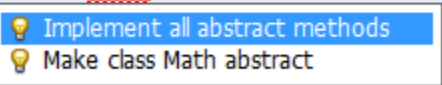


- Make class Force abstract
- Create Subclass

Step 3: Now create Math Class (derived) and inherit Force class, it shows implement all abstract methods.

```
abstract class Force
{
    public abstract void add(int a,int b);
    public abstract void multi(int a,int b);
}

class Math extends Force
{
    // Implement all abstract methods
    // Make class Math abstract
}
```

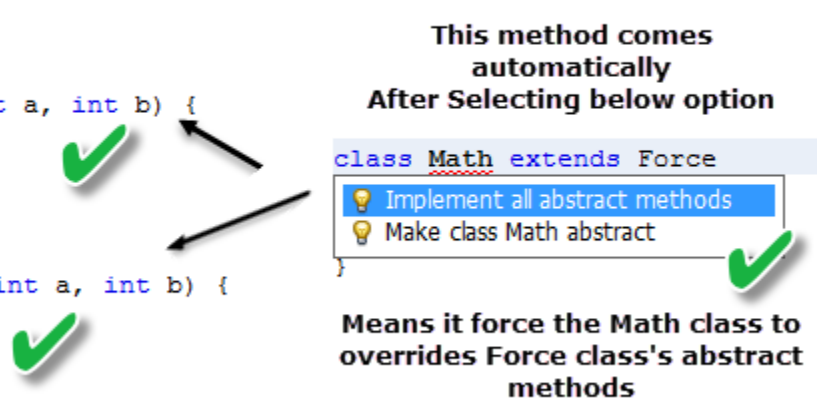


- Implement all abstract methods
- Make class Math abstract

Step 4: See the effects and write the logic of add() and multi() function.

```
class Math extends Force
{
    @Override
    public void add(int a, int b) {
    }

    @Override
    public void multi(int a, int b) {
    }
}
```



This method comes automatically After Selecting below option

```
class Math extends Force
{
    Implement all abstract methods
    Make class Math abstract
}
```

Means it force the Math class to overrides Force class's abstract methods





Step 5: Now create an object of Math class in the main() and implement your logic.

```

abstract class Force
{
    public abstract void add(int a,int b);
    public abstract void multi(int a,int b);
}

class Math extends Force
{
    public void add(int a, int b) {
        System.out.println("A+B = "+ (a+b));
    }

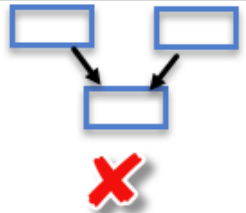
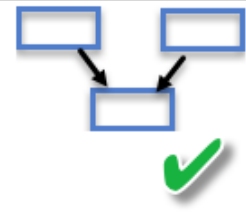
    public void multi(int a, int b) {
        System.out.println("A*B = "+ (a*b));
    }
}

public class JavaApplication1 {
    public static void main(String[] args) {
        Math m1=new Math();
        m1.add(5,6);
        m1.multi(5, 6);
    }
}

```

Interface

- 📄 We know that Java does not support the concept of Multiple Inheritance. Means we cannot override more than one class at a time. But we can implement more than one interface in the class.

| | |
|---|---|
| <pre> class stu { //Logic.... } class marks { //Logic.... } class result extends stu,marks { } </pre> <div style="text-align: center;">  </div> <p>Multiple Inheritance using class is Not possible in Java</p> | <pre> interface if1{ //Logic.... } interface if2 { //Logic.... } class result implements if1,if2 { //Logic.... } </pre> <div style="text-align: center;">  </div> <p>Multiple implementation of interface is possible</p> |
|---|---|



- Interface is similar to class which is collection of final variables and abstract methods.
- An interface follows the concept of has-a relationship. It means one class can implement more than one interface at a time.
- Same as an abstract class, we cannot create an object of an interface.
- Interface contains final data members and abstract methods which does not have a method body. Each method in an interface is also implicitly public and abstract, so there is no need write abstract and public keyword before any method.
- "implements" keyword** is used to create relationship from interface to class.

Syntax#

```
interface interfaceName {  
    ....  
    final datatype data member=value;  
    returntype functionName();  
}
```

Example#

```
interface if1  
{  
    final int x=20;  
    void add(int a,int b);  
}
```

Example# Demo of interface and class**Output View**

```
E:\javaDemo>javac inf1.java  
  
E:\javaDemo>java inf1  
Area of triangle = 10000.0  
Area of rectangle = 20000.0
```

Coding View**interface if1**

```
{  
    void area(double w,double h);  
}
```

class Triangle implements if1

```
{  
    public void area(double w,double h)  
    {  
        System.out.println("Area of triangle = " + (h*w*0.5));  
    }  
}
```

class Rectangle implements if1



```
{
    public void area(double w,double h)
    {
        System.out.println("Area of rectangle = " + (h*w));
    }
}

class inf1
{
    public static void main(String args[])
    {
        Triangle t1=new Triangle();
        Rectangle r1=new Rectangle();
        t1.area(100,200);
        r1.area(100,200);
    }
}
```

Now let see the difference between interface and abstract class

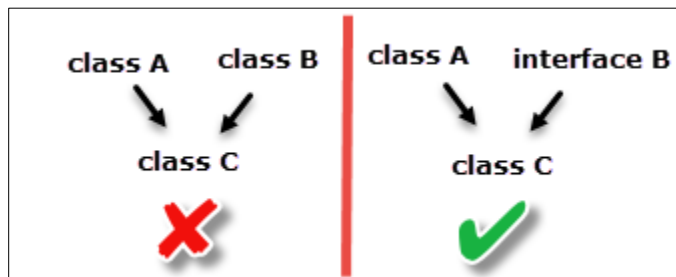
| Interface | Abstract Class |
|---|---|
| Interface does not contain any constructor. | Abstract class can contain constructor. |
| Interface is implemented by a class. | Abstract class is extended by a class. |
| An Interface cannot extend or implement any class. | An Abstract class extend or implements any class. |
| Interface contains only abstract method. | Abstract class contains non abstract method and abstract methods. |
| It support multiple inheritance. | It does not support multiple inheritance. |
| Interface can have only final data members. | Abstract class contains normal data members and final data members. |
| There default access specifier of interface method are public. | The default access specifier of abstract class methods are default. |
| To inherit in derived class "implements" keyword is used. | To inherit in derived class "extends" keyword is used. |
| It represents "has-a" relationship. | It represent "is-a" relationship. |
| Example: Interface if1{Logic..} Interface if2{Logic...} class Math implements if1,if2 { Logic.... } | Example: abstract class abs1 { Logic... } class Math extends abs1 { Logic.... } |

When to use interface?

If a C class is extending a class A and we also want to inherit abstract class B to force method override. But it is not possible as Java does not support multiple inheritance. So instead of



creating B as an abstract class, create an interface, interface B which will implement class C so in a way Class C can inherit class A and interface B. Hence this is a way Java supports multiple inheritance.



Extending or Implementing Inheritance

| | | |
|---|--|---|
| <pre>class abc { //Logic.. } interface if1 extends abc { //Logic... }</pre> | <pre>interface if2 { //Logic.... } interface if1 implements if2 { //Logic... }</pre> | <pre>interface if2 { //Logic.... } interface if1 extends if2 { //Logic... }</pre> |
|---|--|---|

Interface can extend other Interface but cannot implement other interface or extend other class.

Interface Reference

Java permits reference variables creation with interface but not object. Means we cannot create an object of the interface but we can create a reference of Interface like a class.

Whenever we define a new interface, actually we are defining a new reference data type. Then we can use interface name anywhere we can use as other data type name. If the variable is declared as an interface type, it can reference any object of any class that implements the interface.

Example# Interface Reference

Output View

```
E:\javaDemo>javac inf2.java

E:\javaDemo>java inf2
Sum = 30
Sub = 15
X = 22
```

Coding View



```
interface force
{
    final int x=22;
    void add(int a,int b);
    void sub(int a,int b);
}

class math implements force{
    int y=30;
    public void add(int a,int b)
    {
        System.out.println("Sum = " + (a+b));
    }

    public void sub(int a,int b)
    {
        System.out.println("Sub = " + (a-b));
    }

    void mul(int a,int b)
    {
        System.out.println("Mul = " + (a*b));
    }
}

class inf2
{
    public static void main(String args[])
    {
        force f1;
        math m1=new math();
        f1=m1;
        f1.add(10,20);
        f1.sub(20,5);
        System.out.println("X = " + f1.x);
        //f1.mul(5,10) and f1.y both are not accessible
    }
}
```

In above program, We are unable to call mul() data function and y data member with the f1 object, because mul() data function and y data member is not defined in the Force interface. We can only access the data functions or data members defined on the interface through a variable declared as having that interface, even if the implemented class has some new methods. That's why we can call add(), sub() methods and access x data member, but not mul() data function and y data member.

| |
|-------------------------|
| Dynamic Method Dispatch |
|-------------------------|





- C++ implements the concept of runtime polymorphism by the concept of pointer and Virtual function, but Java does not support the concept of pointer. Java implements the concept of run time polymorphism by the Dynamic method dispatch.
- Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- Dynamic the name itself suggests that it is related with run time.
- Dispatch means determining which method to call.
- Dynamic method dispatch means the method that is to be called is determined at runtime.
- Java runtime decides which methods to be called whether base class's method or derived class depends on the object points.
- Dynamic method dispatch also supports concept of upcasting where parent class reference variable refers to a child class object.
- As we have seen in Interface reference, A superclass reference variable can refer to a subclass object. Java uses this concept to resolve calls to override methods at run time.

Example# Demo of Dynamic method dispatch

Output View

```
E:\javaDemo>javac dynamic.java

E:\javaDemo>java dynamic
In base Sub = 8
In derived Sub = 8
Sum = 12
In derived Sub = 8
```

Coding View

```
class base
{
    public void sub(int a,int b)
    {
        System.out.println("In base Sub = " + (a-b));
    }
}

class derived extends base
{
    public void add(int a,int b)
    {
        System.out.println("Sum = " + (a+b));
    }
}
```



```
    }

    public void sub(int a,int b)
    {
        System.out.println("In derived Sub = " + (a-b));
    }
}

class dynamic
{
    public static void main(String args[])
    {
        base b1=new base();
        derived d1=new derived();

        b1.sub(10,2);

        d1.sub(10,2);
        d1.add(10,2);

        b1=d1;
        b1.sub(10,2);
    }
}
```

| |
|-----------------------------------|
| Static binding vs Dynamic binding |
|-----------------------------------|

| Static binding | Dynamic binding |
|---|---|
| Static binding occurs during compilation. | Dynamic binding occurs during runtime. |
| It uses type information to call any method. | It uses object to call any method. |
| Example of static binding is function overloading means during compilation it decides which function to call. | Example of dynamic binding is dynamic method dispatch means at runtime object decides which function to call. |
| It is faster than dynamic binding. | It is slower than static binding. |

System.out.println



System.out.println

System

- System is a final built-in class present in java.lang package.
- The purpose of System class is to provide standard input, output and errors.
- The System class contains several useful class data members and data functions. It cannot be instantiated.

out

- out is a static final variable of System class which is of the type PrintStream.
- Since out is a static variable, it can be called with just the class name "System". System.out. out here denotes the reference variable of the type PrintStream class.

println

- println() is a public method in PrintStream class to print the data values. The println method prints the argument passed to the standard console as new line.