Inheritance:-

The process of getting properties and behaviors from one class to another class is called inheritance.

Properties	:	variables
Behaviors	:	methods

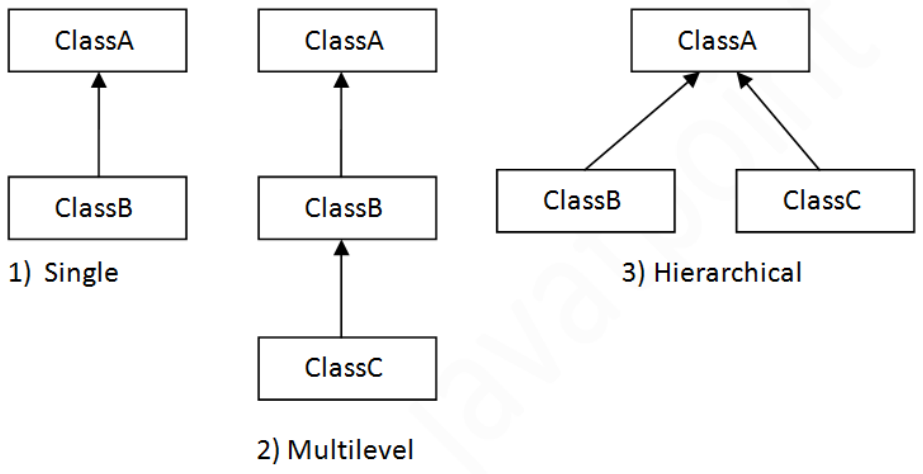
1. The main purpose of the inheritance is code extensibility whenever we are extending automatically the code is reused.
2. The idea behind inheritance in java is that you can create new classes that are built upon existing classes.
3. Inheritance is also known as **is-a** relationship means two classes are belongs to the same hierarchy.
4. By using **extends** keyword we are achieving inheritance concept.
5. In the inheritance the person who is giving the properties is called parent the person who is taking the properties is called child.
6. To reduce length of the code and redundancy of the code sun peoples introducing inheritance concept.

Why use inheritance in java

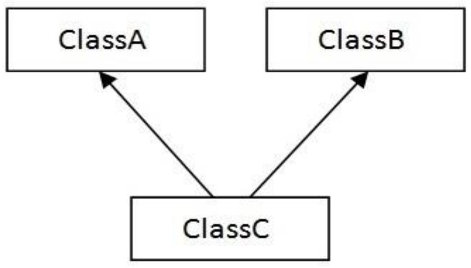
- For Method Overriding (so runtime polymorphism can be achieved).
- For Code Reusability.

Types of inheritance:

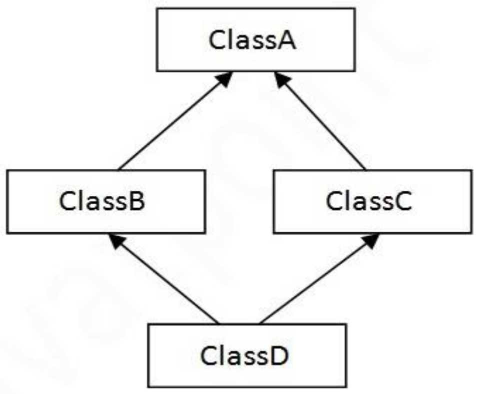
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.In java programming, multiple and hybrid inheritance is supported through interface only.



When a class extends multiple classes i.e. known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

1) Single level

Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.

Single Inheritance example program in Java

```
Class A
{
  public void methodA()
  {
    System.out.println("Base class method");
  }
}

Class B extends A
{
  public void methodB()
  {
    System.out.println("Child class method");
  }
  public static void main(String args[])
  {
    B obj = new B();
    obj.methodA(); //calling super class method
    obj.methodB(); //calling local method
  }
}
```

2) Multi level

Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A.

Multilevel Inheritance example program in Java

```
Class X
{
   public void methodX()
   {
     System.out.println("Class X method");
   }
}
Class Y extends X
{
public void methodY()
{
System.out.println("class Y method");
}
}
Class Z extends Y
{
   public void methodZ()
   {
     System.out.println("class Z method");
   }
   public static void main(String args[])
   {
     Z obj = new Z();
     obj.methodX(); //calling grand parent class method
     obj.methodY(); //calling parent class method
     obj.methodZ(); //calling local method
  }
}
```

3) Multiple

The process of getting properties and behaviors form more than one super class to the one child class. The multiple inheritance is not possible in the java language so one class can extends only one class at time it is not possible to extends more than one class at time.

4) **Hierarchical inheritance:-**

The process of getting properties and behaviors from one super class to the more than one sub classes is called hierarchical inheritance.

5) **Hybrid inheritance:-**

Combination of any two inheritances is called as hybrid inheritance. If are taking the multilevel and hierarchical that combination is called hybrid inheritance.

```
class A
{
        Void m1()  }      parent   class/
        Void m2()  }      super    class/
        Void m3()  }      base class
}

 class B extends A  }
 {                  }    child
        class/ Void m4()  sub
        class       }
         Void m5()  }   derived class
}

Class C extends B
{
Void m6();
}
```

Note 1:-

It is possible to create objects for both parent and child classes.

a. If we are creating object for parent class it is possible to call only parent specific methods.

```
A  a=new
A();
a.m1();
a.m2();
a.m3();
```

b. if we are creating object for child class it is possible to call parent specific and child specific.

```
B b=new B(); b.m1(); b.m2();
b.m3(); b.m4(); b.m5();
```

c. if we are creating object for child class it is possible to call parent specific methods and child specific methods.

```
C  c=new
C();
c.m1();
```

```
                c.m2();
                c.m3();
                c.m4();
                c.m5();
                c.m6();
```

Note:-

1. Every class in the java programming is a child class of object.
2. The root class for all java classes is Object class.
3. The default package in the java programming is java.lang package.

Both declarations are same:-

class Test
{
};
class String
{
};
	class Test extends Object
	{
	};
	class String extends Object
	{
	};

Super:

The **super** keyword in java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Super keyword is used to represent

1) Call the Super class variable.
2) Call the super class constructor.
3) Call the super class methods.

Calling of super class variables:-

Super keyword is not required
class Parent
{
        int a=10;
        int b=20;
};
class Child extends Parent
{
        int x=100;
        int y=200;
        void add**(int i,int j)**
	Super keyword is required
	class Test1
	{
	        int a=10;
	        int b=20;
	};
	class Test extends Test1
	{
	        int a=100;
	        int b=200;
	        void add**(int a,int b)**

{
System.out.println(i+j);
System.out.println(x+y);
System.out.println(a+b);
 	
}
public static void main(String[] args)
{
Child c=new Child();
c.add(1000,2000);
}
};
	{
	System.out.println(a+b);
	
	System.out.println(this.a+this.b);
	System.out.println(super.a+super.b);
	
	}
	public static void main(String[] args)
	{
	Test t=new Test();
	t.add(1000,2000);
	}
	};

Calling of super class methods:-

Super keyword is not required	**Super keyword is required**

class Parent
{
void m1()
{
System.out.println("parent class method");
}
};
class Child extends Parent
{
void m2()
{
m1(); System.out.println("child class method");
}
void m3()
{
m1(); System.out.println("child class method");
m2();
}
public static void main(String[] arhs)
{
Child c=new Child();
c.m3();
}}
	class Parent
	{
	void m1()
	{
	System.out.println("parent class method");
	}
	};
	class Child extends Parent
	{
	void m1()
	{
	System.out.println("child class method");
	}
	void m2()
	{
	this.m1(); System.out.println("child class method");
	super.m1();
	}
	public static void main(String[] arhs)
	{
	Child c=new Child();
	c.m2();}};

Calling of super class constructors:-

Ex 1:- inside the constructors super keyword must be first statement of the constructor otherwise the compiler raise compilation error.

No compilation error

```
class Test1
{
        Test1(int i,int j)
        {
                System.out.println(i);
                System.out.println(j);
System.out.println("two arg constructor");
        }
};
class Test extends Test1
{
        Test(int i)
        {
                super(100,200);
System.out.println("int -arg constructor");

        }
        public static void main(String[] args)
        {
                Test t=new Test(10);
        }
}
```

Compilation error

```
class Test1
{
Test1(int i,int j)
{
System.out.println(i); System.out.println(j);
System.out.println("two arg constructor");
}
};
class Test extends Test1
{
Test(int i)
{
System.out.println("int -arg constructor");
super(100,200);
}
public static void main(String[] args)
{
Test t=new Test(10);
}
}
```

Note: -

- **Inside the constructors it is possible to call only one constructor at a time that calling must be first statement of the constructor.**
- Inside the constructor if we are providing **super** and **this** key words at that situation the Compiler will place the zero argument “super ();” keyword at first line of the constructors.
- If we are declaring any constructor calling then compiler won’t generate any type of keywords.

Strictfp:-

- Strictfp is a modifier applicable for classes and methods.
- If a method is declared as strictfp all floating point calculations in that method will follow IEEE754 standard. So that we will get platform independent results.
- If a class is declared as strictfp then every concrete method in that class will follow IEEE754 standard so we will get platform independent results.

Native:-

- Native is the modifier only for methods but not for variables and classes.
- The native methods are implemented in some other languages like C and C++ hence native methods also known as “foreign method”.
- For native methods implementation is always available in other languages and we are not responsible to provide implementation hence native method declaration should compulsory ends with “;”.

Ex: - public native String intern ();
Public static native void yield ();
Public static native void sleep (long);

Final:-

- Final is the modifier applicable for classes, methods and variables (for all instance, Static and local variables).
- If a class is declared as final, then we cannot inherit that class i.e., we cannot create any child class for that final class.
- Every method present inside a final class is always final by default but every variable present inside the final class need not be final.
- The main advantage of final modifier is we can achieve security as no one can be allowed to change our implementation.
- But the main disadvantage of final keyword is we are missing key benefits of Oops like inheritance and polymorphism. Hence is there is no specific requirement never recommended to use final modifier.

Case 1:-
Ex :-overriding the method is possible.
class Parent
{
void andhra()
{
System.out.println("Andhra Pradesh");
}
}
class Child extends Parent
{
void andhra()
{
	Ex:- overrding the method is not possible
	
	class Parent
	{
	void andhra()
	{
	System.out.println("Andhra Pradesh");
	}
	}
	class Child extends Parent
	{
	final void andhra()
	{

System.out.println("division is not possible");
}
public static void main(String[] args)
{
Child c=new Child();
c.andhra();
}
};
	System.out.println("division is not possible");
	}
	public static void main(String[] args)
	{
	Child c=new Child();
	c.andhra();
	}
	};

Case 2:-
child class creation is possible
 	
class Parent
{
};
class Child extends Parent
{
};
	child class creation is not possible getting compilation error.
	
	final class Parent
	{
	};
	class Child extends Parent
	{
	};

Case 3:-
Reassignment is possible.
 	
class Test
{
public static void main(String[] args)
{
int a=10; a=a+10;
System.out.println(a);
}
};
	For the final variables reassignment is not possible.
	class Test
	{
	public static void main(String[] args)
	{
	final int a=10; a=a+10;
	System.out.println(a);
	}
	};

Note:-
Every method present inside a final class is always final by default but every variable present inside the final class need not be final.

```
final class Test
{
```

```
int a=10;
void m1()
{
System.out.println("m1 method is final");
System.out.println(a+10);
}
public static void main(String[] args)
{
Test t=new Test();
t.m1();
}
}
```

Polymorphism is one of the OOPs features that allow us to perform a single action in different ways. For example,
Let’s say we have a class Animal that has a method sound(). Since this is a generic class so we can’t give it a implementation like: Roar, Meow, Oink etc. We had to give a generic message.

As you can see that although we had the common action for all subclasses sound() but there were different ways to do the same action. This is a perfect example of polymorphism (feature that allows us to perform a single action in different ways). It would not make any sense to just call the generic sound() method as each Animal has a different sound. Thus we can say that the action this method performs is based on the type of object.

What is polymorphism in programming?

Polymorphism is the capability of a method to do different things based on the object that it is acting upon. In other words, polymorphism allows you define one interface and have multiple implementations. As we have seen in the above example that we have defined the method sound() and have the multiple implementations of it in the different-2 sub classes. Which sound() method will be called is determined at runtime so the example we gave above is a **runtime polymorphism example**.

Types of polymorphism

1. Method Overloading in Java – This is an example of compile time (or static polymorphism)
2. Method Overriding in Java – This is an example of runtime time (or dynamic polymorphism)

Static Polymorphism

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java that allows a class to have more than one constructor having different argument lists.

Three ways to overload a method
In order to overload a method, the argument lists of the methods must differ in either of these:

1. Number of parameters.

For example: This is a valid case of overloading
add(int, int)
add(int, int, int)

2. Data type of parameters.

For example:
add(int, int)
add(int, float)

3. Sequence of Data type of parameters.
For example:
add(int, float)
add(float, int)

Invalid case of method overloading:

When I say argument list, I am not talking about return type of the method, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

int add(int, int)
float add(int, int)

Method overloading:-

Ex:-

```
class Test
{
void m1(char ch)
{
System.out.println(" char-arg constructor ");
}
void m1(int i)
{         Overloaded Methods
System.out.println("int-arg constructor ");
}
void m1(int i,int j)
{
System.out.println(i+"------"+j);
}
public static void main(String[] args)
{
Test t=new Test();
t.m1('a');
 t.m1(10);
t.m1(10,20);
}
}
```

Method Overloading and Type Promotion
When a data type of smaller size is promoted to the data type of bigger size than this is called type promotion, for example: byte data type can be promoted to short, a short data type can be promoted to int, long, double etc.
What it has to do with method overloading?
Well, it is very important to understand type promotion else you will think that the program will throw compilation error but in fact that program will run fine because of type promotion.
Lets take an example to see what I am talking here:

```
class Demo{
   void disp(int a, double b){
         System.out.println("Method A");
   }
   void disp(int a, double b, double c){
         System.out.println("Method B");
   }
   public static void main(String args[]){
         Demo obj = new Demo();
         /* I am passing float value as a second argument but
```

```
            * it got promoted to the type double, because there
            * wasn't any method having arg list as (int, float)
            */
           obj.disp(100, 20.67f);
   }
}
```

Interfaces

1. Interface is also one of the type of class it contains only abstract methods.
2. For the interfaces also .class files will be generated.
3. Each and every interface by default abstract hence it is not possible to create an object.
4. Interfaces not alternative for abstract class it is extension for abstract classes.
5. 100 % pure abstract class is called interface.
6. The Interface contains only abstract methods means unimplemented methods.
7. Interfaces give the information about the functionalities it is not giving the information about internal implementation.
8. To provide implementation for abstract methods we have to take separate class that class we can call it as implementation class for that interface.
9. Interface can be implemented by using implements keyword.
10. For the interfaces also the inheritance concept is applicable.
11. An interface can extend multiple interfaces but a class cannot extend multiple classes.
12. Extends and implements both keywords can be used in same line of code but extends should be first keyword then implements.
13. An interface can be declare inside the class is called nested interface.
14. An interface can be declare inside the another interface is called nested interface.

Syntax:-
Interface interface-name
Ex: - interface it1

Note: -
- If we are declaring or not by default interface methods are public abstract.

```
Interface it1                         abstract interface it1
{                                     {
 Void m1();                Both are same            public abstract void m1();
 Void m2();                                         public abstract void m2();
 Void m3();                                         public abstract void m3();
 }                                                  }
```

Example program:-

```
Interface it1
{
Void m1();
Void m2(); Void m3();
}
Class Test implements it1
{
Public void m1()
{
System.out.println("m1-method implementation ");
}
Public void m2()
{
System.out.println("m2-method implementation");
}
Public void m3()
{
System.out.println("m3 –method implementation");
}
Public static void main(String[] args)
{
Test t=new Test();
t.m1();
t.m2();
t.m3();
}
}
```

Adaptor class:-

It is a intermediate class between the interface and user defined class. And it contains empty implementation of interface methods.

Limitation of interface	**advantage of adaptor classes**

interface it
{
void m1();
void m2();
.
.
 	
.
void m100()
}
Class Test implements it
{
Must provide implementation of 100 methods otherwise compiler raise compilation error
}
	interface it
	{
	void m1();
	void m2();
	.
	.
	.
	void m100()
	}
	
	class Adaptor implements it
	{
	void m1(){}
	void m2(){}
	;
	;
	void m100(){}
	}
	class Test implements it
	{
	must provide the 100 methods implementation
	};
	class Test extends Adaptor
	{
	provide the implementation of required methods.
	};