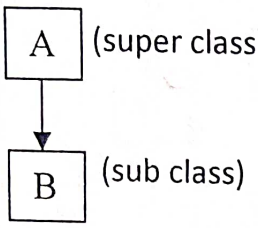
Inheritance: Extending the properties of one class into another is known as inheritance.

Type of Inheritance:

There are three types of Inheritance in Java language as given below.

1. **Single inheritance**



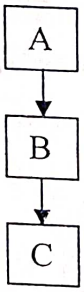
A class from which properties are inherited is known as super class and the class in which the properties are inherited is known as sub class.

When one class inherits properties from other class then such type of inheritance is known as single inheritance.

In the above diagram class A is super class and class B is sub class of super class A and therefore class B contains all the properties of class A as well as properties of its own class.

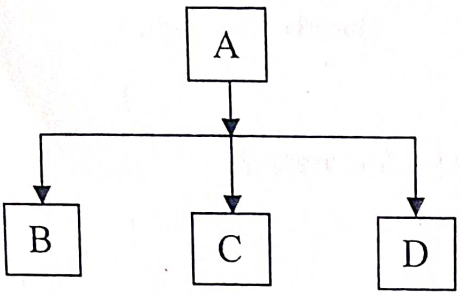
2. **Multilevel Inheritance**

When one class inherits properties from another class and another class inherits properties from next class and so on then such type of inheritance is known as multilevel inheritance.



In the above diagram, class C is sub class of super class B and class B is sub class of super class A hence class C contains all the properties of super classes.

3. Hierarchical Inheritance:-



When multiple classes extends properties from single class then such type of inheritance is known as hierarchical inheritance. In the above diagram class B, C and D are sub classes of super class A, therefore all the subclasses contain properties of super classes and the properties of their own class.

extends keyword

extends is standard inbuilt keyword of Java language, it is used to extends the properties of super class in sub class. Whenever any subclass extends the properties of super class then subclass will get all the properties of super class except private properties.

Que. Explain method overriding with suitable example?

Ans:

Whenever same prototype of method is present in super class and sub class then method of sub class overrides the method of super class, such type of method is known as overridden method and the technique is known as method overriding. In method overriding, object of sub class can access overridden method of sub class and cannot directly access overridden method of super class. The concept of method overriding is useful to change the logical instructions of method of existing class without changing the existing code.

Example 1:

```
class A
{
        public void show()
        {
                System.out.println("I am in A");
        }
}
```

```
class B extends A
{
        public void show()
        {
                System.out.println("I am in B");
        }
}

class UseMe
{
        public static void main(String args[])
        {
                B b1=new B();
                b1.show();
        }
}
```

Output: I am in B

In the above example, class B is sub class of super class A and both the classes contain same prototype of show () method with different logical instructions therefore show () method of class B will override the show () method of class A and object of class B will access show () method of its own class i.e. show () method of class A is hidden for the object of sub class B. show () method is known as overridden method.

Que. Explain super keyword and super method?

Ans:

super is standard inbuilt keyword of java language. It is used to access member variable, methods and constructors of super class from the block of sub class i.e. super keyword provides three different features as follows.

1. To access member variable of super class in sub class.
2. To access constructor of super class in sub class.
3. To access methods of super class in sub class.

Example 1:

```
class A
{
        public void show()
        {
                System.out.println("I am in A");
        }
}

class B extends A
{
        public void show()
        {
                System.out.println("I am in B");
        }
}

class UseMe
{
        public static void main(String args[])
        {
                B b1=new B();
                b1.show();
        }
}
```

Output:

I am in B

In the above example, class B is sub class of super class A and both the classes contain same prototype of show () method due to which class B will not directly access the show () method of super class A. To access the show () method of super class A, we have to use super keyword that helps to access members of super class from the block of sub class as given below.

Example 2:

```
class A
{
        public void show()
        {
                System.out.println("I am in A");
        }
}

class B extends A
{
        public void show()
        {
                super.show();
                System.out.println("I am in B");
        }
}

class UseMe
{
        public static void main(String args[])
        {
                B b1=new B();
                b1.show();
        }
}
```

Output:

I am in A

I am in B

In the above example, super keyword is used to access the show () method of super class A.

Que. Explain the flow the execution of constructor in inheritance?

Or

Que. Explain super constructor of inheritance?

Ans:

Whenever the object is created, it will allocate memory as per the number of member variables available in its class as well as in the super classes and therefore it is necessary to initialize the value of all the member variables of instance. To initialize the value of member variables of object, java interpreter invoke all the default constructor of super classes and after the complete execution of all the default constructors of super classes it will invoke constructor of its own class (requested constructor).

Example1:

```
class A
{
        private int n;
        public  A()
        {
                a=10;
        }
        public void showA()
        {
                System.out.println("A="+a);
        }
}

class B extends A
{
        private int b;
        public B(int n)
        {
        b=n;
        }
        public void showB()
```

```
        {
                System.out.println("B="+b);
        }
}

class UseMe
{
        public static void main(String arg[])
        {
                B b1=new B(200);
                b1.showA();
                b1.showB();
        }
}
```

output:

A=10

B=200

To change the flow of constructor i.e. to invoke parameterized constructor of super class from the block of constructor of sub class, we have to use super keyword as shown below.

Example1:

```
class A
{
        private int n;
        public  A()
        {
                a=10;
        }
        public  A(int n)
        {
                a=n;
        }
        public void showA()
```

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

class B extends A
{
    private int b;
    public B(int n1,int n2)
    {
    super(n1);
    b=n2;
    }
    public void showB()
    {
        System.out.println("B="+b);
    }
}

class UseMe
{
    public static void main(String arg[])
    {
        B b1=new B(100,200);
        b1.showA();
        b1.showB();
    }
}
```

Output:

```
A=100
B=200
```

Que. Explain super member variable of java language?

Ans:

Whenever super and sub class contain same variable/object then variable/object of sub class will hide the variable/object of super class such type of variable/object is known as hidden variable or hidden object, in such case, sub class will not able to access member variable of super class.

Example 1:

```
class A
{
        public  int n;
        public void showA()
        {
                System.out.println("A="+n);
        }
}

class B extends A
{
        public int n;
        public void setB(int n1,int n2)
        {
                n=n1;
                n=n2;
        }
        public void showB()
        {
                System.out.println("B="+n);
        }
}
```

```
class UseMe
{
        public static void main(String args[])
        {
                B b1=new B();
                b1.setB(10,20);
                b1.showA();
                b1.showB();
         }
}
```

Output:

A=0

B=20

To access the member variable of super in sub class we have to use *super keyword as shown* below.

Example 2:

```
class A
{
        public  int n;
        public void showA()
        {
                System.out.println("A="+n);
        }
}

class B extends A
{
        public int n;
        public void setB(int n1,int n2)
        {
                super.n=n1;
                n=n2;
```

```
    }
    public void showB()
    {
            System.out.println("B="+n);
    }
}

class UseMe
{
    public static void main(String args[])
    {
        B b1=new B();
        b1.setB(10,20);
        b1.showA();
        b1.showB();
    }
}
```

Output:

```
A=10
B=20
```

final keyword

final is standard inbuilt keyword of java language. It provides three different features in java language i.e. final variable, final method and final class.

Que. Explain final variable of java language with suitable example?

Ans:

Whenever any variable or object of class is declared as final then that variable or object is known as final variable or final object. It is always necessary to initialize the value of final variable at the time of initialization and once it is initialized, compiler will not allow changing the value of final variable or final object. It is used to define the constant value as shown below.

Example1:

```
class Example1
{
        public  static void main(String args[])
        {
                final int a=10;
                System.out.println(a);
                a=20;
                System.out.println(a);
         }
 }
```

In the above example, variable 'a' is declared as final and therefore compiler will not allow to change the value of variable 'a' hence above program will generate following error.

Error: can't assign value to final variable 'a'

a=20;

Example2:

```
class Example2
{
        public  static void main(String args[])
        {
                final int a=10;
                System.out.println(a);
                int b=a*2;
                System.out.println(b);
         }
 }
```

Output:

```
10
20
```

Que. Explain final method of Java language?

Ans:

Whenever any method of class is declared as final then that method is known as final method. final method is considered as the last version method of class in which it is declared. During the compilation process, compiler will not allow sub class to override the final method of super class.

Example1:

```
class A
{
        final public void show()
        {
         System.out.println("I am in A");
         }
  }

   class B extends A
   {
           public void show()
           {
           System.out.println("I am in B");
           }
    }

     class UseMe
     {
           public static void main(String args[])
           {
           B b1=new B();
           b1.show();
           }
      }
```

In the above example, class B is subclass of super class A overriding the final method show() of super class A but compiler will not allow to override the final method show() of super class A in sub class B and hence above program will generate following error.

Error: Can't override the final method show() in sub class A

Que. Explain the final class of java language?

Ans: Whenever any class is declared as **final** then that class is known as final class. **final** class does not allow to extend the properties in sub class i.e. compiler will not allow sub class to extend the properties of **final** class but we can access the properties of **final** class by creating the object of final class.

Example1:

```
final class A
{
        public void showA()
        {
        System.out.println("I am in A");
        }
}

class B extends A
{
        public void showB()
        {
        System.out.println("I am in B");
        }
}
```

Error: Can't extend the final class A in class B

Example2:

```
final class A
{
        public void showA()
        {
```

```
        System.out.println("I am in A");
        }
}

class UseMe
{
        public  static void main(String args[])
        {
        A a1=new A();
        a1.showA();
        }
  }
```

output: I am in A

abstract keyword

abstract is standard inbuilt keyword of java language. It provides two different features i.e. abstract class and abstract method.

Que. Explain abstract class with suitable example?

Ans.

Whenever any class is declared as abstract then that class is known as abstract class. we cannot create the object of abstract class but we can access the properties of abstract class by extending the properties of abstract class. Any class that extends the properties of abstract class is known as concrete class.

Example1:

```
abstract class A
{
        public  void show()
        {
        System.out.println("I am in A");
        }
 }
```

```
class UseMe
{
        public static void main(String args[])
        {
        A a1=new A();
        a1.show();
        }
}
```

Error : Can't create the object of abstract class A.

In the above example, since class A is an abstract class therefore main program cannot create the object of abstract class A. But we can access the properties of abstract class A by extending the properties of abstract class A as shown below.

```
abstract class A
{
        public  void show()
        {
        System.out.println("I am in A");
        }
}

class B extends A
{
        public  void show1()
        {
        System.out.println("I am in B");
        }
}

class UseMe
{
        public static void main(String args[])
        {
```

```
        B b1=new B();
        b1.show();
        b1.show1();
        }
}
```

output:

```
I am in A
I am in B
```

Que. Explain abstract method with suitable example?

or

Que. Explain dynamic dispatch method with suitable example?

Ans.

Whenever any prototype of method is declared as abstract then that method is known as abstract method provided abstract method should be present in abstract class. Any subclass that extends the properties of abstract class is known as concrete class and it is necessary for the concrete class to implement the body of abstract method of super abstract class with explicit access specifier as public.

Dynamic dispatch method is the concept of java language in which object of concrete class dispatches its methods to the object of abstract class and object of abstract class can access all those methods of concrete class that are declared as abstract in abstract class.

Example1:

```
abstract class A
{
        public abstract void show();
}

class B extends A
{
        public  void show()
        {
        System.out.println("I am in B");
        }
```

```
}

class Magic
{
        public  static void showMagic(A a)
        {
        a.show();
        }
  }

   class UseMe
   {
          public static void main(String args[])
          {
                  B b1=new B();
                  Magic.showMagic(b1);
           }
     }
```

output: I am in B

In the above example, class B is concrete class of super class (abstract) A containing the implemented body of abstract method show() of abstract class A. Since the class B is concrete class of abstract class A therefore object of class A can hold the instance of class B and can access show() method. During the execution of above program, execution control will dispatch the abstract methods of class B to the object of class A.