INHERITANCE

Need of Inheritance:

- Inheritance enables code reusability and saves time.
- Inheritance is used to declare characteristics of classes inheriting it,without giving its implementation.
- It is one of the most important concepts of OOPS.

Inheritance:

★ Inheritance can be defined as the process where one class acquires the properties (**methods and fields**) of another class.

★ The class which inherits the properties of other class is known as subclass. It's also called a derived **or child class** and the class whose properties are inherited is known as superclass (**base or parent class**).

The following program is the simple example for Compile time polymorphism:

```
class A
{
   int add(int a, int b)
   {
       return a+b;
   }
   int  add(int a, int b, int c)
   {
       return a+b+c;
   }
}
public class B
{
  public static void main(String args[])
```

```
  {
     A obj = new A();
     System.out.println(obj.add(10, 20));
     System.out.println(obj.add(10, 20, 30));
  }
}
```

Output:

30

60

The following program is the simple example for Run time polymorphism:

```
class A
{
     public void myMethod()
     {
       System.out.println("Overridden Method");
      }
}
public class B extends A
{
    public void myMethod()
    {
        System.out.println("Overriding Method");
    }
    public static void main(String args[])
    {
        B obj = new B();
        obj.myMethod();
  }
}
```

Output:

Overriding Method

extends keyword**:**

The extends is a Java keyword, which is used in the inheritance process of Java. It specifies the superclass in a class declaration using extends keyword. There are two types:

- The class from which another class is derived is called the superclass
- The derived class (the class that is derived from another class) is called a subclass.

Syntax:

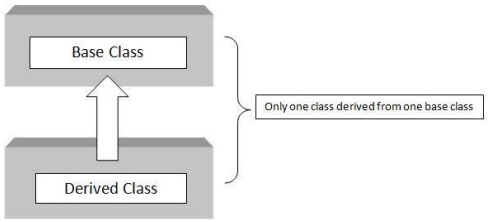
```
class base_class

{
  .....
  .....
}
class derived_class extends base_class
{
  .....
  .....
}
```

Types of Inheritance:

- Single Level Inheritance
- Multiple Inheritance
- Multi-Level Inheritance
- Hierarchical Inheritance
- Hybrid Inheritance

Single inheritance:

When a class extends another one class then only we call it as 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.



Example 1 : Accessing the data members of single inheritance

```
class A
{
    int a = 6;
}
class B extends A
{
    void area()
     {
      System.out.println("The area of the square is … "+a*a);
     }
}
class Main
{
     public static void main(String args[])
    {
        B b1 = new B();
        b1.area();
     }
}
```

Output:

36

Example 2 : Accessing the member function of a single inheritance

```
class A
{
      void methodA(){
          System.out.println("Base class method");
      }
}
class B extends A
{
      void methodB(){
          System.out.println("Child class method");
      }
}
class Main
{
      public static void main(String args[])
      {
          B b1 = new B();
          b1.methodA();            //calling super class method
          b1.methodB();            //calling local method
      }
}
```

Output:

Base class method

Child class method

EXPLANATION:

We Created a Base Class **A** and Using **extends** keyword we create a derived class **B.** By Creating an Object to Class **B(**child class), We Can access Data member and member function of Class **A(Parent class)** . Here we access the member function (**methodA()**) of Class A using Object of Class **B.**

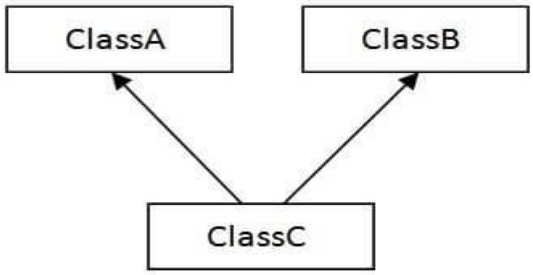
Note:

- We can access data member or member function of the Parent class by using object created to derived class (**Class B**) Or Object created to Base class (**Class A**)

- We cannot access data member or member function of the derived class (**Class B**), Using object of parent class (**Derived Class inherited the property of Base Class**)

Multiple Inheritance:

- ❖ Object Oriented Programming provides a user the feature of multiple inheritance, wherein a class can inherit the properties of more than a single parent class.Java doesn't support multiple inheritance but it can be achieved indirectly through the usage of interfaces.
- ❖ Here the derived class C inherits the features of the base classes A and B through the keyword Interface and Implement.



Interface and Implement Keyword:

- The interface keyword is used to declare a special type of class that only contains abstract methods. To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends).
- The implements keyword is used to implement an interface .

Example 1: for accessing the data members using the keyword interface and implement:

```
interface A
 {
       int a=10;
 }
interface B
 {
       int b=5;
  }
class C implements A,B
{
       int c;
      public void sum()
       {
          c=a+b;
          System.out.println(c);
       }
}
class Main
{
    public static void main(String args[]){
          C c1= new C();
          c1.sum();
    }
 }
```

Output:

15

Example 2: for accessing the member functions using the keyword interface and implement:

```
interface A
  {
    void cricket();
```

```
  }
interface B
  {
     void hockey();
   }
class C implements A,B
{
     public void cricket()
     {
          System.out.println("There are more fans for cricket ");
      }
     public void hockey()
     {
          System.out.println("There are less fans for Hockey");
      }
}
class Main
{
     public static void main(String args[])
{
         C c1= new C();
         c1.cricket();
         c1.hockey();
     }
}
```

Output:

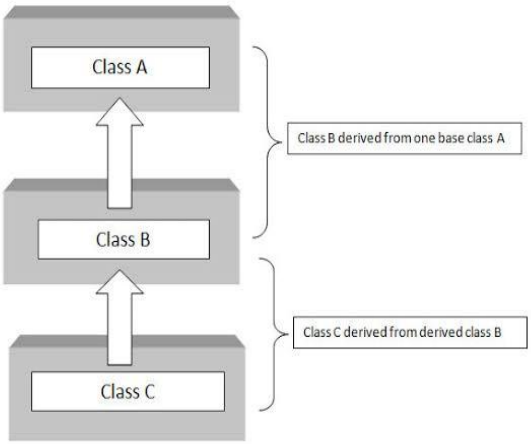
There are more fans for cricket
There are less fans for Hockey

EXPLANATION:

We have created a base classes (A&B) and derived class C by using a keyword **implement**.Because using extend keyword will extend only one base class, so that we using interface and implement keyword to extend more than one class.

Multi-Level Inheritance:

In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below diagram, class A serves as a base class for the derived class B, which in turn class B serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent's members.



Example:

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

```
class  C extends B
{
      public  void methodC()
     {
        System.out.println("class  C method");
      }
}
class  Main
{
     public  static void main(String args[])
    {
         C  obj = new C();
         obj.methodA();              //calling grand parent class  method
         obj.methodB();             //calling parent class method
         obj.methodC();             //calling local method
     }
}
```

Output:

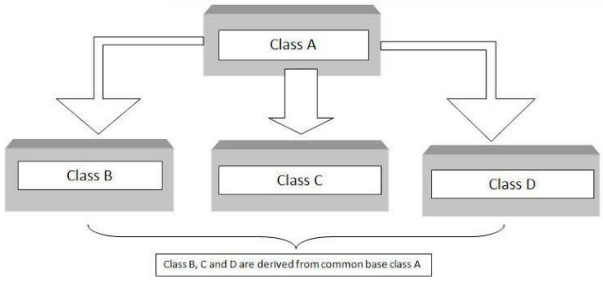
class A method
class B method
class C method

EXPLANATION:

Here a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class.In above,when we call a method obj.methodX(),first the control goes to class **C** which extends class **B,** on turn it extends class **A**...so the function get executed.

Hierarchical Inheritance:

When a class has more than one child classes (sub classes) or in other words more than one child classes have the same parent class then this type of inheritance is known as hierarchical inheritance.



Example:

```
class A
{
    public void methodA()
    {
        System.out.println("method of Class A");
    }
}
class B extends A
{
    public void methodB()
    {
        System.out.println("method of Class B");
    }
}

class C extends A
{
    public void methodC()
    {
        System.out.println("method of Class C");
    }
}
```

```
class D extends A
{
   public void methodD()
   {
      System.out.println("method of Class D");
   }
}

class Main
{
   public static void main(String args[]) {
       B obj1 = new B();
      C obj2 = new C();
      D obj3 = new D();
     //All classes can access the method of class A
      obj1.methodA();
      obj2.methodA();
      obj3.methodA();
   }
}
```

Output:

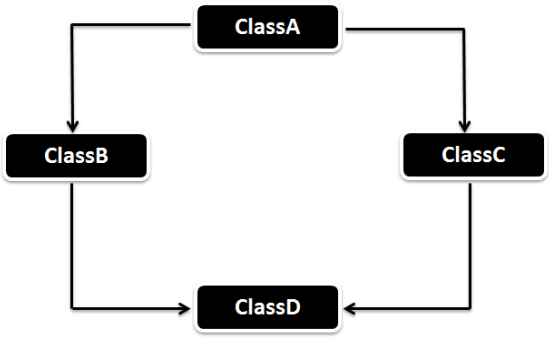
method of Class A
method of Class A
method of Class A

EXPLANATION:

Here all derived classes belong to (i.e extends) the same base class,so that All classes can access the method of class A.

Hybrid Inheritance:

Hybrid Inheritance is a combination of two types of inheritances.so,for this we taking hierarchical Inheritance and Multiple Inheritance. Since in Java Multiple Inheritance is not supported directly we can achieve Hybrid inheritance also through Interfaces only.



As we can see in the above diagram **ClassA** is the Parent for both **ClassB** and **ClassC** which is hierarchical Inheritance and again **ClassB** and **ClassC** again act as Parent for **ClassC**(Multiple Inheritance which is not supported by Java).

Example 1: Accessing the data members of hybrid inheritance:

```
interface A
{
   int a = 10;
}

interface B extends A
{
   int b = 5;
}

interface C extends A
{
   int c = 20;
}
class D implements B, C
{
    int total;
    public void sum()
    {
        total = a + b + c;
```

```
        System.out.println(total);
    }
}
class Main
{
    public static void main(String args[])
    {
        D d1 = new D();
        d1.sum();
    }
}
```

Output:

35

Explanation:

- In this program,
  interface B & C extends interface A --- Hierarchical inheritance
  class D implements interface B, C --- Multiple inheritance

Example 2: Accessing the member functions of hybrid inheritance:

```
interface A
{
   public void methodA();
}

interface B extends A
{
   public void methodB();
}

interface C extends A
{
   public void methodC();
```

```
}
class D implements B, C
{
    public void methodA()
    {
        System.out.println("Calling method A");
    }
   public void methodB()
    {
        System.out.println("Calling method B");
    }
   public void methodC()
    {
        System.out.println("Calling method C");
    }
}
class Main
{
    public static void main(String args[])
    {
        D d1 = new D();
        d1.methodA();
        d1.methodB();
        d1.methodC();
    }
}
```

Output:

Calling method A
Calling method B
Calling method C

Explanation:

- In this program,
  → interface B & C extends interface A --- Hierarchical inheritance
  → class D implements interface B, C --- Multiple inheritance