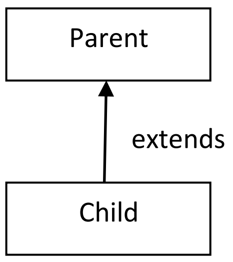
INHERITANCE

Inheritance:

- **Inheritance in java** is a mechanism in which one object acquires all the properties and behaviors of parent object.
- One class acquiring or inheriting the properties of another class is called inheritance.
- The idea behind inheritance in java is that you can create new classes that are built upon existing classes.
- When we inherit from an existing class, we can reuse methods and fields of parent class, and add new methods and fields also.
- Inheritance represents the **IS-A relationship**, also known as *parent-child* relationship.
- Using inheritance we can achieve code reusability and method overriding.
- A class that is inherited is called a *Super class or Parent class or Base class.*
- The class that does the inheriting is called a *Sub class or Child class or Derived class*
- In java inheritance can be achieved by using “extends” keyword.
- General form of inheritance is

```
class Parent
{
    // members of parent
}
class Child extends Parent
{
     /*members of
   Child + members of parent*/
}
```



- The **extends keyword** indicates that ,making a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.
- Once Child class extends Parent class we can access members of Parent class by object of Child class.
- In the above general form members inside Parent class are inherited into Child class.
- Here members mean “variables” and “methods” inside a class.

Example:

```
class Parent
{
   public void m1()
   {
      System.out.println("parent method m1");
   }
}
```

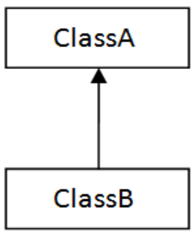
```
class Child extends Parent
{
   public void m2()
   {
      System.out.println("child method m2");
   }
}
class InheritDemo
{
   public static void main(String args[])
   {
      // creating object for Child class
       Child c=new Child();
       c.m1();
       c.m2();
     //creating object for parent class
       Parent p=new Parent();
       p.m1();// we can only access m1 but not m2.


   }
}
```

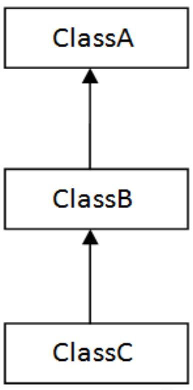
Output:

```
     parent method m1
     child method m2
     parent method m1
```

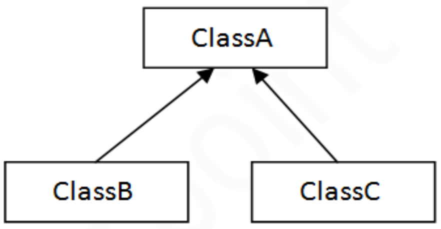
- In java we have different types of inheritance
  - Simple inheritance (or) Single inheritance
  - Multi level inheritance
  - Multiple inheritance (not supported at class level)
  - Hierarchical inheritance
- **Note:** In java Multiple inheritance is not supported at class level but it can be achieved by using interfaces.



1) Single



2) Multilevel



3) Hierarchical

Simple inheritance:

- Only one class acquiring the properties of another class.
- **Example:**

```
class A
{
   public void m1()
   {
      System.out.println("A class method m1");
   }
}
class B extends A
{
   public void m2()
   {
      System.out.println("B class method m2");
   }
}
class SimpleInheritance
{
   public static void main(String args[])
   {
       B b=new B();
       b.m1();
       b.m2();
   }
}
```

Output:

```
A class method m1
B class method m2
```

Multilevel inheritance:

```
class A
{
   public void m1()
   {
      System.out.println("A class method m1");
   }
}
class B extends A
{
   public void m2()
   {
      System.out.println("B class method m2");
   }
}
```

```
class C extends B
{
   public void m3()
   {
      System.out.println("C class method m3");
   }
}
class MultiLevelInheritance
{
   public static void main(String args[])
   {
       C c=new C();
       c.m1();
       c.m2();
       c.m3();
   }
}
```

Output:

```
A class method m1
B class method m2
C class method m3
```

Hierarchical inheritance:

```
class A
{
   public void m1()
   {
      System.out.println("A class method m1");
   }
}
class B extends A
{
   public void m2()
   {
      System.out.println("B class method m2");
   }
}
class C extends A
{
   public void m3()
   {
      System.out.println("C class method m3");
   }
}
class HierarchicalInheritance
{
   public static void main(String args[])
   {
       B b=new B();
       C c=new C();
```

```
          b.m1();
          b.m2();
          c.m1();
          c.m3();
      }
  }
```
Output:

```
  A class method m1
  B class method m2
  A class method m1
  C class method m3
```

Why multiple inheritance is not supported in java?

- To reduce the complexity and simplify the language, multiple inheritance is not supported in java at class level.
- Consider a scenario where A, B and C are three classes. The C class inherits A and B classes.
- If A and B classes have same method and you call it from child class object, there will be ambiguity to call method of A or B class.
- Since compile time errors are better than runtime errors, java renders compile time error if you inherit 2 classes.
- So whether you have same method or different, there will be compile time error now.

```
class A
{
     void msg()
     {
        System.out.println("Hello");
     }
}
class B
{
     void msg()
     {
        System.out.println("Welcome");
     }
}
class C extends A,B
{//suppose if it were
    public Static void main(String args[])
     {
       C obj=new C();
       obj.msg();//Now which msg()method would be invoked?

     }
}
```

****

Method Overriding:

- If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in java.
- In other words, If subclass provides the specific implementation of the method that has been provided by one of its parent class, it is known as method overriding.
- **Usage of Java Method Overriding**
    1. Used to provide specific implementation of a method that is already provided by its super class.
    2. Used for runtime polymorphism
- **Rules for Java Method Overriding**
    1. Method must have same name as in the parent class
    2. Method must have same parameter as in the parent class.
    3. must be IS-A relationship (inheritance).
    4. Method can't be static

Example:

```
class Bank
{
   // ROI means Rate Of Interest
   int getROI()
   {
        return 0;
   }
}
class SBI extends Bank
{
   int getROI()
   {
        return 8;
   }
}
class ICICI extends Bank
{
   int getROI()
   {
        return 7;
   }
}
class OverrideBank
{
   public static void main(String args[])
   {
      SBI s=new SBI();
      ICICI i=new ICICI();
      System.out.println("SBI ROI(%):"+s.getROI());
      System.out.println("ICICI ROI(%):"+i.getROI());
   }
}
```

Output:

```
SBI ROI(%):8
ICICI ROI(%):7
```

****

super keyword:

- 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.
- 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.

To refer immediate parent class instance variable:

```
class Parent
{
     String s;
}
class Child extends Parent
{
    String s;
    void show(String s1,String s2)
     {
        super.s=s1;
        s=s2;
        System.out.println("Parent string s: "+super.s);
        System.out.println("Child string s: "+s);
     }
}
class SuperMemberDemo
{
     public static void main(String args[])
     {
          Child c=new Child();
          c.show("java","programming");
     }
}
```

Output:

```
     Parent string s: java
     Child string s: programming
```

To refer immediate parent class method:

```
class Parent
{
     void m1()
     {
        System.out.println("this is parent method m1()");
     }
}
```

```
class Child extends Parent
{
   void showm1()
   {
     //calling Parent m1 inside showm1() using super
     super.m1();
   }
}
class SuperMethodDemo
{
     public static void main(String args[])
     {
          Child c=new Child();
          c.showm1();
     }
}
```

Output :

```
this is parent method m1()
```

To invoke immediate parent class constructor:

```
class Parent
{
   Parent(String s)
   {
     System.out.println("Hello from Parent: "+s);
   }
}
class Child extends Parent
{
    Child(String s1)
     {
        super(s1);
     }
}
class SuperTest
{
    public static void main(String args[])
     {
         Child ch=new Child("Java");
     }
}
```

Output:

```
     Hello from Parent: Java
```

****

final keyword:

- 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:

- If you make any variable as final, you cannot change the value of final variable (It will be constant).
- Example of final variable,
  There is a final variable speed limit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike
{
    final int speedlimit=90;//final variable
    void run()
   {
         speedlimit=400; // not allowed to change
    }
    public static void main(String args[])
   {
         Bike obj=new  Bike();
         obj.run();
    }
}//end of class
```

Output: `Compile Time Error`

final method:

- If you make any method as final, you cannot override it.
- Example of final method

```
class Bike
{
  final void run()
     {
        System.out.println("running");
     }
}
class Honda extends Bike
{
   void run()// not allowed to override
   {
     System.out.println("running safely with 100kmph");
   }
   public static void main(String args[])
   {
      Honda h= new Honda();
      h.run();
   }
}
```

Ouput: `Compile time error`

final class:

- If you make any class as final, you cannot extend it.
- Example of final class

```
final class Bike
{
}
class Honda1 extends Bike // not allowed to extend
{
  void run()
  {
      System.out.println("running safely with 100kmph");
  }
  public static void main(String args[])
  {
      Honda1 h= new Honda1();
      h.run();
  }
}
```

Ouput: `Compile time error`

****