4.Inheritance

❖ Inheritance

- ➢ Inheritance is a mechanism to derive a new class from old class.
- ➢ New class is a sub class or child class.
- ➢ Old class is a base class or super class, parent class.
- ➢ Define a sub class

```
class sub name extends super class
{
    Variable declaration;
    Function(Method) declaration
}
```

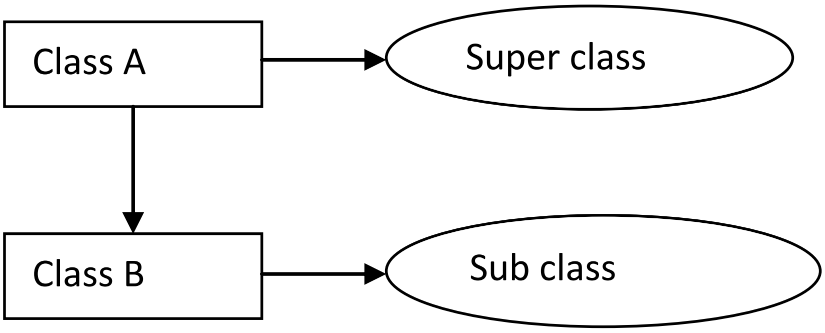
- ➢ The keyword extends signifies that the properties of super class name are extends to the sub class name.
- ➢ **Advantages:**
  - Reusability
  - Extensibility
  - Data hiding
  - Overriding
- ➢ **Disadvantages:**
  - Both classes are tightly coupled.
  - They cannot work independently of each other.
  - Changing code in super class method also affects subclass functionality.

❖ Types of Inheritance:

- Single Inheritance
- Multilevel Inheritance
- Hierarchical Inheritance
- Multiple Inheritance
- Hybrid Inheritance

❖ Single Inheritance

- ➢ Sub class can be deriving from super class this mechanism is known as single inheritance.



- ➢ **Syntax:**

```
Class A
{
    - - - -
        - - - -
}
```

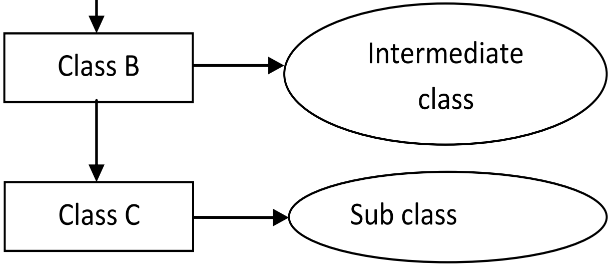
```
        Class B extends A
        {
            - - - -
            - - - -
        }
```

- **Example:** Program for single inheritance.

```
class A
{
        void add()
        {
                int a, b, c;
                a=2;
                b=4;
                System.out.println("Value C :"+ (a+b));
        }
}
class B extends A
{
        void mul()
        {
                int  a, b, c;
                a=2;
                b=4;
                System.out.println("Value C :"+ a*b);
        }
}
class singleinheritance
{
        public static void main(String[] args)
        {
                B ob=new B();
                ob.add();
                ob.mul();
        }
}
```

❖ Multilevel Inheritance

- C class is derived from B class & B class is derived from A class this type of mechanism is called multilevel inheritance.
- Sub class can be derived from another super class.



- **Syntax:**

```
class A
{
  - - - -
   - - - -
}
class B extends A
{
     - - - -
     - - - -
 }
 class C extends B
 {
     - - - -
     - - - -
  }
  class  multilevel
  {
    - - - -
    - - - -
   }
```

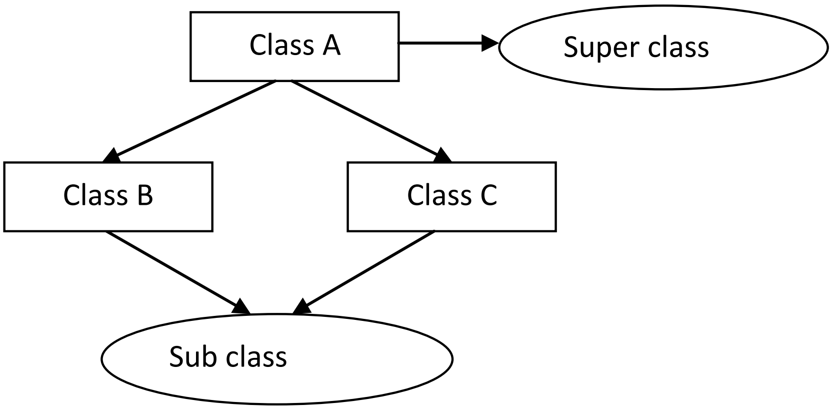
- **Example:** Program for multilevel inheritance.

```
class A
{
        void add()
        {
         int a, b, c;
         a=2;
         b=4;
         System.out.println("Value C :"+ a+b);
        }
}
class B extends A
{
        void mul()
        {
         int  a, b, c;
         a=2;
         b=4;
         System.out.println("Value C :"+ a*b);
        }
}
class C extends B
{
        void div()
        {
         int  a, b, c;
         a=4;
         b=2;
         System.out.println("Value C :"+ a/b);
        }
}
```

```
class Multilevel
{
                public static void main(String[] args)
                {
                  C ob=new C();
                  ob.add();
                  ob.mul();
                  ob.div();
                }
}
```

❖ Hierarchical Inheritance

- One super class and more than one sub class it's called hierarchical inheritance.



- **Syntax:**

```
class A
  {
       - - - -
       - - - -
  }
  class B extends A
  {
        - - - -
     - - - -
  }
  class C extends A
  {
         - - - -
      - - - -
   }
   class hierarchical
   {
        - - - -
     - - - -
    }
```

- **Example**:-Program for hierarchical inheritance.

```
class A
{
    void add()
    {
      int a, b, c;
      a=2;
      b=4;
      System.out.println("Value C :"+ a+b);
    }
}
class B extends A
{
    void mul()
    {
      int  a, b, c;
      a=2;
      b=4;
      System.out.println("Value C :"+ a*b);
    }
}
class C extends A
{
    void div()
    {
      int  a, b, c;
      a=4;
      b=2;
      System.out.println("Value C :"+ a/b);
    }
}
class hierarchical
{
    public static void main(String[] args)
    {
      B ob1=new B();
      ob1.add();
      ob1.mul();
      C ob2=new C();
      ob2.div();
    }
}
```

❖ Super keyword:

- ➢ The super is a keyword defined in the java programming language.
- ➢ Super is a reference variable that is used to refer parent/super/base class objects.
- ➢ Super is a used to access variables, constructors and methods of a parent/super/base class.
- ➢ **Example** to use the super keyword to call the constructor of the superclass:

```
public class car
{
        int speed=200;
}
class bike extends car
{
        int speed=100;
        void dispaly()
        {
              System.out.println(super.speed);//super to access speed method of car class
```

```
                        System.out.println(speed);
                    }
            }
            class vehicle
            {
                    public static void main(String args[])
                    {
                            bike ob=new bike();
                            ob.display();
                    }
            }
```

- The syntax **super.method_name()** is used to call a method of the super class from the sub class.
-
```
class car
{
        car()
        {
                System.out.println("Printed in car class.");
        }
}
class bike extends car
{
        bike()
        {
                super();        // here super keyword is to access constructor of car(super) class
                System.out.println("Printed in bike class");
        }
}
class vehicle
{
        public static void main(String[] args)
        {
                bike ob = new bike();
        }
}
```

  - The syntax **super()** is used to call a constructor of the super class from the sub class.

❖ When Constructors Are Called in Multilevel hierarchy

- The answer is that in a class hierarchy, constructors are called in order of derivation, from superclass to subclass.
- Further, since super( ) must be the first statement executed in a subclass' constructor, this order is the same whether or not super( ) is used.
- If super( ) is not used, then the default or parameter less constructor of each superclass will be executed. The following program illustrates when constructors are executed:
- **Demonstrate when constructors are called.**

```
        class A
        {
            A()
            {
                System.out.println("Inside A's constructor.");
            }
        }
```

```
// Create another subclass by extending A.
class B extends A
{
    B()
    {
        System.out.println("Inside B's constructor.");
    }
}
class C extends B
{
    C()
    {
        System.out.println("Inside C's constructor.");
    }
}
class CallingCons
{
    public static void main(String args[])
    {
        C c = new C();
    }
}
```

❖ Method overriding

- When method in sub class has same name as a method in super class then method in sub class is said to override the method in super class.
- **Advantage of Method Overriding**
  - Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
  - Method Overriding is used for Runtime Polymorphism
- **Rules for Method Overriding:**
  - method must have same name as in the parent class
  - Method must have same parameter as in the parent class.
- **Example 1:** Program for overriding

```
class Superclass
{
        void print()
        {
                System.out.println("Printed in Superclass.");
        }
}
class Subclass extends Superclass
{                                   Overriding
        void print() ←──────────    method
        {
                super.print();
                System.out.println("Printed in Subclass");
```

```
    }
    public static void main(String[] args)
    {
        Subclass s = new Subclass();
        s.print();
    }
}
```

- **Example 2:** Program for overriding

```
class A
{
    int i,j;
    A(int a,int b)
    {
        i=a;
        j=b;
    }
    void show()
    {
        System.out.println("i and j:"+i+” “+j);
    }
}
class B extends A
{
    int k;
    B(int a,int b,int c)
    {
        super(a,b);
        k=c;
    }
    void show()
    {
        super.show();
        System.out.println("k:"+k);
    }
}
class overriding
{
    public static void main(String[] args)
    {
        B ob=new B(10,20,30);
        ob.show ( );
    }
}
```

❖ What is difference between overloading and overriding?

Method overloading	Method overriding
When two or more methods in a class have the same method names with different arguments, it is said to be method overloading.	When a method in a subclass has the same method name with same arguments as that of the super class, it is said to be method overriding.

Overloading does not block inheritance from the super class.	Overriding blocks inheritance from the super class.
In case of method overloading, parameter must be different.	In case of method overriding parameter must be same.

- Basically overloading and overriding are different aspects of polymorphism.
  - Static/early binding polymorphism: **overloading**
  - dynamic/late binding polymorphism: **overriding**

❖ Java Abstract Class

- An abstract class is a class that is declared by using the abstract keyword.
- We have seen that by making a method final we ensure that the method is not redefined in a subclass.
- That is, the method can never be subclass.
- Java allows us to do something that is exactly opposite to this.
- That is, we can indicate that a method must always be redefined in a subclass, thus making overriding compulsory.
- This is done using the modifier keyword abstract in the method definition.
- **Example:**

```
abstract   class   shape
{
    ……………
    …………….
    abstract void draw ( );
    ……………..
    …………….
}
```

- When a class contains one or more abstract methods, it should also be declared abstract as shown in the example above.
- While using abstract classes, we must satisfy the following conditions:
  - We cannot use abstract classes to insatiate objects directly.
  - For example ,

    shape s= new shape ( ); is illegal because shape is an abstract class.
  - The abstract methods of an abstract class must be defined in its subclass.
  - We cannot declare abstract constructors or abstract static methods.
- **Example:**

```
        abstract class One
        {
                int i;
                int j;
                One(int a, int b)
                {
                        i = a;
                        j = b;
                }
                abstract int area();
        }
        class Two extends One
        {
                Two(int a, int b)
                {
```

```
                super(a, b);
            }
            int area()
            {
                System.out.println("Inside Area for.");
                return i * j;
            }
        }
        class AbstractDemo
        {
            public static void main(String args[])
            {
                // One f = new One(10, 10); // illegal now
                Two t = new Two(9, 5);
                One o; // this is OK, no object is created
                o = t;
                System.out.println("Area is " + o.area());
            }
        }
```

❖ Using final with Inheritance

- The keyword final has three uses.
  - it can be used to create the equivalent of a named constant.
  - Two uses of final apply to inheritance. Both are examined here.
- **Using final to Prevent Overriding**
  - Using final to Prevent Overriding While method overriding is one of Java's most powerful features
  - There will be times when you will want to prevent it from occurring. To disallow a method from being overridden, specify final as a modifier at the start of its declaration.
  - Methods declared as final cannot be overridden.
  - The following fragment illustrates final:

```
class A
{
    final void show()
    {
        System.out.println("This is a final method.");
    }
}
class B extends A
{                               ERROR! Can't
    void show()                 override.
    {
        System.out.println("Illegal!");
    }
}
```

  - Because show( ) is declared as final, it cannot be overridden in B.
  - If you attempt to do so, a compile-time error will result.
  - Methods declared as final can sometimes provide a performance enhancement.
- **Using final to Prevent Inheritance**
  - Sometimes you will want to prevent a class from being inherited.

- To do this, precede the class declaration with final.
- Declaring a class as final implicitly declares all of its methods as final, too.
- As you might expect, it is illegal to declare a class as both abstract and final since an abstract class is incomplete by itself and relies upon its subclasses to provide complete implementations.
- Here is an example of a final class:

  ```
  final class A
  {
      // ...
  }
  // The following class is illegal.
  class B extends A
  {
      // ERROR! Can't subclass A
      // ...
  }
  ```

- As the comments imply, it is illegal for B to inherit A since A is declared as final.