# OOPS

## Oops concepts:-

## 1. Inheritance

## 2. Polymorphism

## 3. Abstraction

## 4. Encapsulation.

## Inheritance.

- ❖ The process of acquiring **fields**(variables) and **methods**(behaviors) from one class to another class is called **inheritance**.
- ❖ The main ***objective*** of inheritance is code *extensibility* whenever we are extending class automatically the code is reused.
- ❖ In inheritance one class giving the properties and behavior & another class is taking the properties and behavior.
- ❖ Inheritance is also known as **is-a** relationship. By using **extends** keyword we are achieving inheritance concept.
- ❖ extends keyword used to achieve inheritance & it is providing *relationship* between two classes .
- ❖ In java **parent** class is *giving properties to child class* and **Child** is acquiring properties from Parent.
- ❖ To reduce length of the code and redundancy of the code sun people introduced inheritance concept.

**Application Code before Inheritance vs After inheritance**

- ❖ **With Child Object we can call parent props/methods( summary) Simple examples**

# Types Of Inheritance

There are five types of inheritance in java

*1. Single inheritance*

*2. Multilevel inheritance*

*3. Hierarchical inheritance*

*4. Multiple inheritance*

*5. Hybrid Inheritance*

## *Single inheritance:-*

➢ One class has one and only one direct super class is called single inheritance.

➢ In the absence of any other explicit super class, every class is implicitly a subclass of Object class**.**

## *Multilevel inheritance:-*

❖ One Sub class is extending Parent class then that sub class will become Parent class of next extended class this flow is called multilevel inheritance.
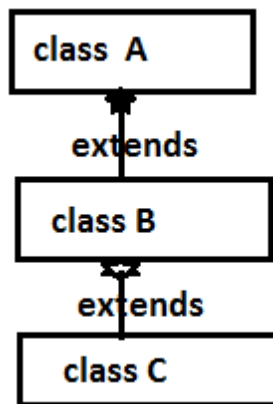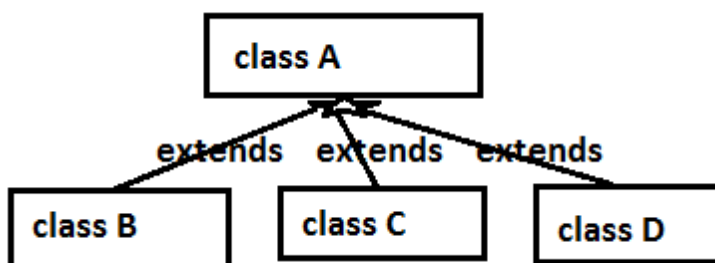
```
class  A
   ↑
 extends
class B
   ↑
 extends
class C
```

```
Child.java    *Parent.java ⊠    Application.java
  1  package com;
  2
  3  public class Parent {
  4⊖     void m1() {
  5            System.out.println("I am parent class m1 method");
  6        }
  7  }
  8
  9  class Children extends Parent{
 10⊖     void m2() {
 11            System.out.println("I am Children class m2 method");
 12        }
 13  }
 14  class GrandChildren extends Children{
 15⊖     void m3() {
 16            System.out.println("I am GrandChildren class m2 method");
 17        }
 18  }
 19
```

## Hierarchical inheritance :-

➢ *More than one sub class is extending single Parent is called hierarchical inheritance.*



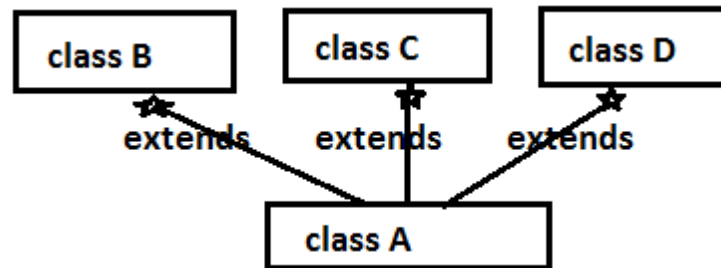## Example Code

```
Child.java    *Parent.java ⊠    Application.java
  1  package com;
  2
  3  public class Parent {
  4⊖     void m1() {
  5            System.out.println("I am parent class m1 meth
  6        }
  7  }
  8
  9  class Children extends Parent{
 10⊖     void m2() {
 11            System.out.println("I am Children class m2 me
 12        }
 13  }
 14  class GrandChildren extends Parent{
 15⊖     void m3() {
 16            System.out.println("I am GrandChildren class
```

- *Class Children and GrandChildren both extending **Parent** class*

## Multiple inheritance:-

➢ One *sub class* is extending **more than one super class** is called Multiple inheritance and java not supporting multiple inheritance because it is creating ambiguity problems (confusion state) .

➢ As Java not supporting multiple inheritance hence in java one class able to extends only one class at a time but it is not possible to extends more than one class.
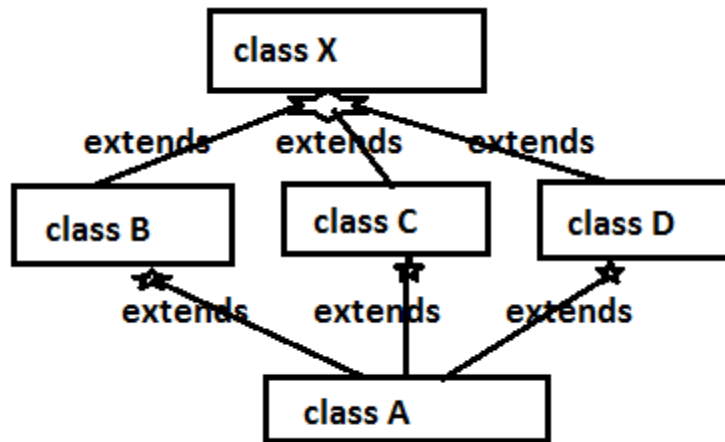
```
┌──────────┐   ┌──────────┐   ┌──────────┐
│ class B  │   │ class C  │   │ class D  │
└──────────┘   └──────────┘   └──────────┘
      ↖            ↑              ↖
   extends      extends        extends
           ┌──────────────┐
           │   class A    │
           └──────────────┘
```

## Example Code

```
 1  package com;
 2
 3⊖ public class A extends  C, B{
 4
 5⊖     void m2() {
 6
 7      }
 8
 9  }
10
11  class B {
12
13⊖     void m1() {
14          System.out.println("I am B class M1 method");
15      }
16
17  }
18
19  class C {
20⊖     void m1() {
21          System.out.println("I am B class M1 method");
22      }
23
24  }
25
```

## Hybrid inheritance:-

➢ Hybrid is combination of any two or more inheritances.

## Preventing inheritance:-

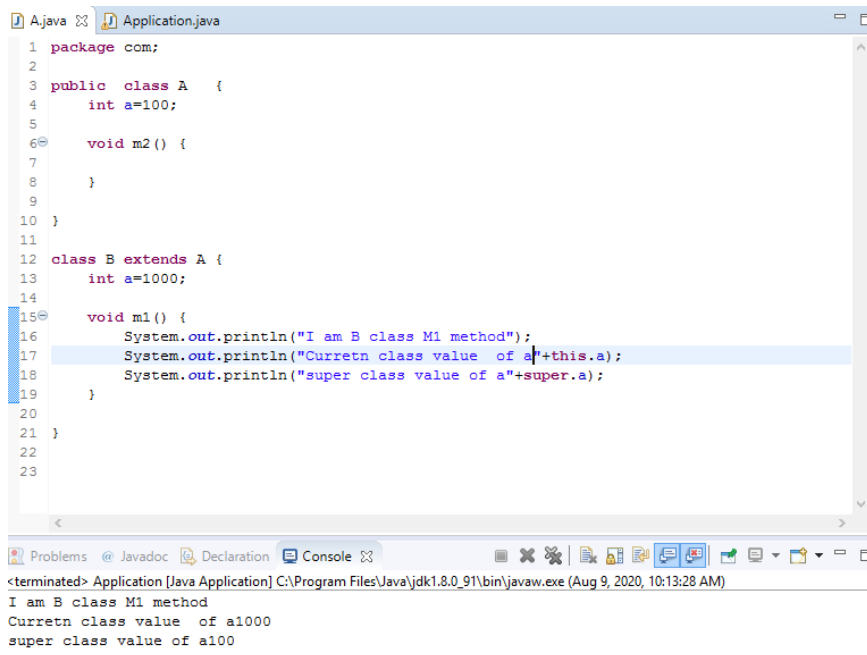➢ You can prevent sub class creation by using **final** keyword in the *parent class declaration.*



*Need to discuss on HAS-A relation .*

## Super keyword:-

**Super** keyword is holding *super class object*. And it is representing

➢ *Super class variables*
➢ *Super class methods*
➢ *Super class constructors.*
➢ **Current class** *variables/methods/constructors are called with* this *variable where as* parent class *or super class variables/methods/constructors are called with* super keyword.
➢ *If both variables having same name in both the class then we will use* this*, * super *to segregate it.*

```
  A.java 23    Application.java
  1  package com;
  2
  3  public   class A   {
  4      int a=100;
  5
  6⊖     void m2() {
  7
  8      }
  9
 10  }
 11
 12  class B extends A {
 13      int a=1000;
 14
 15⊖     void m1() {
 16          System.out.println("I am B class M1 method");
 17          System.out.println("Curretn class value  of a"+this.a);
 18          System.out.println("super class value of a"+super.a);
 19      }
 20
 21  }
 22
 23
```

  Problems   @ Javadoc   Declaration   Console 23
<terminated> Application [Java Application] C:\Program Files\Java\jdk1.8.0_91\bin\javaw.exe (Aug 9, 2020, 10:13:28 AM)
I am B class M1 method
Curretn class value  of a1000
super class value of a100

## *Polymorphism:-*

➢   Polymorphism is the ability of an object to take on many forms
   ➢ Polymorphism shows same functionality(method name same) with
     different logics execution.
   ➢ The ability to appear in more forms is called polymorphism.
   ➢ Polymorphism is a Greek word **poly** *means* **many** and **morphism** means
     **forms**.
     There are two types of polymorphism in java
   ➢ Compile time polymorphism / static binding / early binding
     *[method execution decided at compilation time]*
     *Example :- method overloading.*
   ➢ Runtime polymorphism /dynamic binding /late binding.
   ➢ *[Method execution decided at runtime].*
     *Example :- method overriding.*

   ➢ If we want achieve overloading concept one class is enough.
   ➢ It is possible to overload any number of methods in single java class.

## *Compile time polymorphism [Method Overloading]:-*

➢ If java class allows two or more  **methods with same name** but *different
  number of arguments* such type of methods are called overloaded
  methods.

- ➢ 2) We can overload the methods in two ways in java language
    - a. By passing different number of arguments to the same methods.
        *void m1(int a){}*
        *void m1(int a,int b){}*
    - b. Provide the same number of arguments with different data types.
        *void m1(int a){}*
        *void m1(char ch){}*
        - ➢ **If we want achieve overloading concept one class is enough.**
        - ➢ *It is possible to overload any number of methods in single java class.*

<mark>*Types of overloading:-*</mark>
- ➢ *Method overloading explicitly by the programmer*
- ➢ *Constructor overloading.*

# Runtime polymorphism [Method Overriding]:-
- ➢ If we want to achieve method overriding we **need two class** with parent and child relationship.
- ➢ The parent class method contains some implementation (logics).
        a. If child is satisfied use parent class method.
        b. If the child class not satisfied (required own implementation) then override the method in Child class.
- ➢ A subclass has the same method as declared in the super class it is known as method overriding.
- ➢ The **parent** class method is called ===> **overridden method**
- ➢ The **child** class method is called ===> **overriding method**

*While overriding methods must fallow these rules:*-
- ➢ *While overriding child class method signature & parent class method signatures must be same otherwise we are getting compilation error.*
- ➢ *The return types of overridden method & overriding method must be same.*
- ➢ *While overriding the methods it is possible to maintains same level permission or increasing order but not decreasing order, if you are trying to reduce the permission compiler generates error message "**attempting to assign weaker access privileges** ".*
- ➢ *You are unable to override final methods. (Final methods are preventing overriding)*
- ➢ *Static methods are bounded with class hence we are unable to override static methods.*

➢ If a subclass defines a static method with the same signature as a static method in the superclass, then the method in the subclass *hides* the one in the superclass.

## Parent class reference variable is able to hold child class object but Child class reference variable is unable to hold parent class object.

### Abstraction:-
There are two types of methods in java based on signature.
**a. Normal methods**
**b. Abstract methods**

#### Normal methods:- (component method/concrete method)
➢ Normal method is a method which contains method **declaration** as well as method **implementation**.
***Example:-***
*void m1()* ***--->method declaration***
*{*
*body; **--->method implementation***
*}*
#### Abstract methods:-
➢ The method which is having **only method declaration** but *not method implementations* such type of methods are called **abstract Methods**.
➢ In java every abstract method must end with ";".
*Example : - **abstract void m1 ();** ----> method declaration*

*Based on above representation of methods the classes are divided into two types*
**1) Normal classes.**
**2) Abstract classes.**
#### Normal classes:-
➢ Normal class is a ordinary java class it contains only normal methods .
***Example:-***
*class Test **//normal class***
*{ void m1(){body;} **//normal method***
#### Abstract class:-
**If any abstract method inside the class that class must be abstract.**

### . *Abstract modifier:-*

➢ Abstract modifier is applicable for **methods** and **classes** but not for variables.

➢ To represent particular class is abstract class and particular method is abstract method to the compiler use **abstract** modifier.

➢ The abstract class contains declaration of abstract methods , it says abstract class partially implemented class hence for **partially implemented classes object creation is not possible.**

➢ If we are trying to create object of abstract class compiler generate error message "class is abstract con not be instantiated".

➢ *Abstract class may contains abstract methods or may not contains abstract methods* but object creation is not possible*.*

# Interfaces

➢ **Interface is also one of the type of class it contains only abstract methods.** And Interfaces not alternative for abstract class it is extension for abstract classes.

➢ The abstract class contains **at least one abstract** method but the interface contains **only abstract methods.**

➢ Interfaces giving the information about the **functionalities** and these are not giving the information about internal **implementation**.

➢ Inside the source file it is possible to declare any number of interfaces. And we are declaring the interfaces by using interface keyword.

➢ Interface contains abstract method, for these abstract methods provide the implementation in the **implementation** classes.

➢ Implementation class is nothing but the class that implements particular interface.

➢ While providing implementation of interface methods that implementation methods must be public methods otherwise compiler generate error message "*attempting to assign weaker access privileges".*

### *Marker interface :-*

➢ An interface that has **no members** (methods and variables) is known as marker interface or tagged interface or ability interface.

➢ In java whenever our class is implementing marker interface our class is getting some capabilities .

*Ex:- **serializable** , Cloneable , RandomAccess…etc*

➢ ***By default** the methods which are in interface are public abstract.*

➢ ***The interface contains constants and these constants by default public static final.***

➢ ***For the interfaces** object creation is not possible.*

➢ ***Difference between abstract classes & interfaces?***

| Abstract Class | interfaces |
|---|---|
| The purpose of abstract class is to specify default functionality of an object and let its sub classes explicitly implement that functionality. | It is providing complete abstraction layer and it contains only declarations of the project then write the implementations in implementation classes. |
| An abstract class is a class that declared with **abstract** modifier | Declare the interface by using **interface** keyword. |
| The abstract allows declaring both **abstract & concrete methods**. | The interface allows declaring **only abstract methods**. |
| The abstract class is able to provide implementations of interface methods | The interface contains abstract methods write the implementations in implementation classes. |
| One java class is able to extends only one abstract class at a time. | One java class is able to implements multiple interfaces at a time |
| Inside abstract class it is possible to declare constructors | Inside interface it is not possible to declare methods and constructors. |
| It is not possible to instantiate abstract class. | It is not possible to instantiate Interfaces also. |

***Encapsulation:-***
The process of binding the data and code as a single unit is called encapsulation.

We are able to provide more encapsulation by taking the private data(variables) members.

To get and set the values from private members use getters and setters to set the data and to get the data.

### *Public modifier:-*

 *Public modifier is applicable for variables,methods,classes.*

 *All packages are able to access public members.*

### *Default modifier:-*

 It is applicable for variables,methods,classes.

 We are able to access default members only within the package and it is not possible to access

outside package .

 Default access is also known as package level access.

 The default modifier in java is default.

### *Private modifier:-*

 private modifier applicable for methods and variables.

 We are able to access private members only within the class and it is not possible to access even

in child classes.

### *Protected modifier:-*

 Protected modifier is applicable for variables,methods.

 We are able access protected members with in the package and it is possible to access outside

packages also but only in child classes.

 But in outside package we can access protected members only by using child reference. If wetry

to use parent reference we will get compile time error.