**Polymorphism in java** is a concept by which we can perform a *single action by different ways*. Eg.: going to Mumbai either by old highway or by new highway. Polymorphism is derived from 2 greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So, polymorphism means many forms.

* Static Polymorphism / Method **Overloading** / Compile time Polymorphism
* Dynamic Polymorphism / Method **Overriding** / Run time Polymorphism

**VVVIMP** :

1. Parent class reference can't access child class members and methods **[**this is not applicable in OVERRIDING**]**. And, for OVERRIDING : Jya class cha object tya class chi method call hote.
2. Parent class reference can hold parent object and child object. Child class can hold object of child class, but CAN NOT hold object of parent.
3. **Static** methods : child can access parent class’s static method by creating child’s object or directly using childClassName.methodName
4. **Private** methods : We cannot override private methods. We cannot access private method of parent using child class object. We can overload private methods in Java as a normal method but, you can access these from the same class.
5. **Final :** We can overload a final method, and its possible in JAVA. But we cannot override final methods.
6. **Constructor** : In case of overriding, if parent and child class both possess constructor implementation, and we execute ‘parent newobj = **new** Chilld();’ this statement, then here 1st PARENT constructor and then CHILD constructor will be executed. Both will be executed not one.
7. **Constructor Overriding and Inheritance** : Java class **Does not allow** constructor Overriding and constructor inheritance. But, constructor overloading is allowed.

Variations in **Overloading** a Method:

* Number of parameters passed
* Data type of parameters
* Sequence of data type of parameters
* **Static** : we can overload static method as normal methods.(we can call static methods in main directly without object OR we can call it with object as well. Both things work.)
* **Return type** is not considered (it can be same or different) : if two exact same methods in same class with different return types, then compile time error occurs and compiler ask us to change any one method’s name. If we change parameters here in this case, then error goes away since java consider it as 2 different methods i.e. overloaded methods.
* **Ex:**
* void sum (int a , int b);
* void sum (int a , int b, int c);
* void sum (float a, double b);

**Method Overriding**

Method overriding is: when one of the methods in the super class is **redefined** in the sub-class. In this case, the signature of the method remains the same. **Return type** of overridden methods **can be different.** Overriding : Jya class cha object tya class chi method call hote. We cant override private method.

**Ex:**

class X

{

public int sum()

{

// some code

}

}

class Y extends X

{

public int sum()

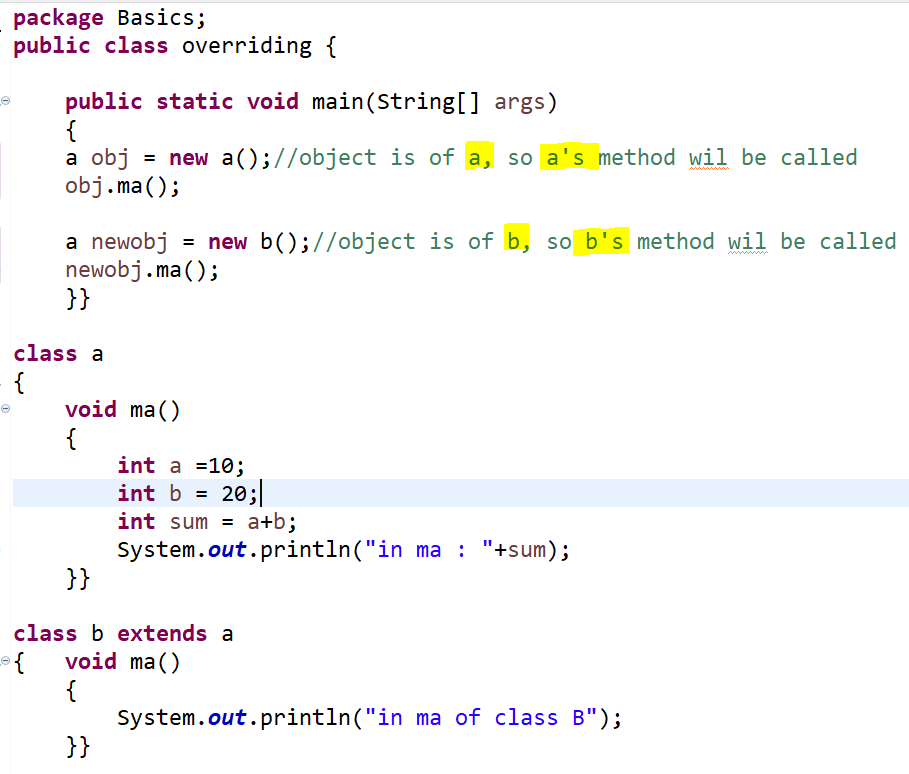
{

//overridden method

//signature is same

}

}



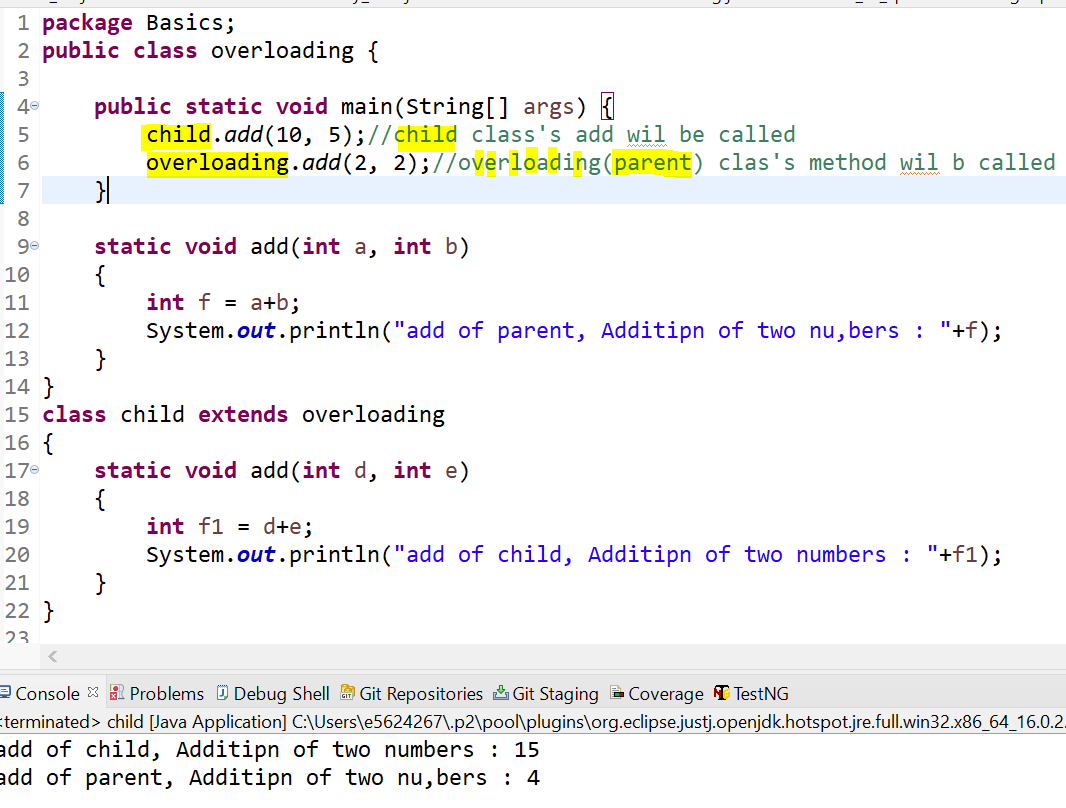
**IMP** : An overloaded method **may or may not have different return types**. But return type alone is not sufficient for the compiler to determine which method is to be executed at run time. Method Overloading means to have two or more methods with same name in the same class with different arguments.

# **What is method hiding in Java and how to use it?**

When super class and sub class contains same method including parameters and if they are static. Then the method in the super class will be hidden by the one that is in the sub class. This mechanism is known as method hiding. in method hiding, it calls method based on the reference not based on the object

We can not override the static methods in a derived class because static methods are linked with the class, not with the object. It means when we call a static method then JVM does not pass this reference to it as it does for all non-static methods. Therefore run-time binding cannot take place for static methods.

Example :



**Effect of exception on overloading and overriding** :

 **If the superclass method does not declare an exception**

* If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.

 **If the superclass method declares an exception**

* If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.