**OPP (Inheritance): Is-A relationship**

The main advantage of is-a relationship is code reusability. By using extends keyword we are going to implement is-a relationship.

public class Test{  
 public static void main(String[] args) {  
 P p = new P();  
 p.m1();  
 //p.m2(); Invalid  
   
 C c = new C();  
 c.m2();  
 c.m1();  
   
 P p1 = new C();  
 p1.m1();  
 //p1.m2() parent class reference can't be used for child class method  
   
 //C c1 = new P(); //Invalid  
 }  
}  
class P{  
 public void m1(){  
 System.*out*.println("Parent");  
 }  
}  
class C extends P{  
 public void m2(){  
 System.*out*.println("Child");  
 }  
}

conclusion:

what ever method parent has by default available to the child and hence on child reference we can call both parent and child.

Whatever method child has by default not available to the parent and hence on parent reference we can’t call child specific methods.

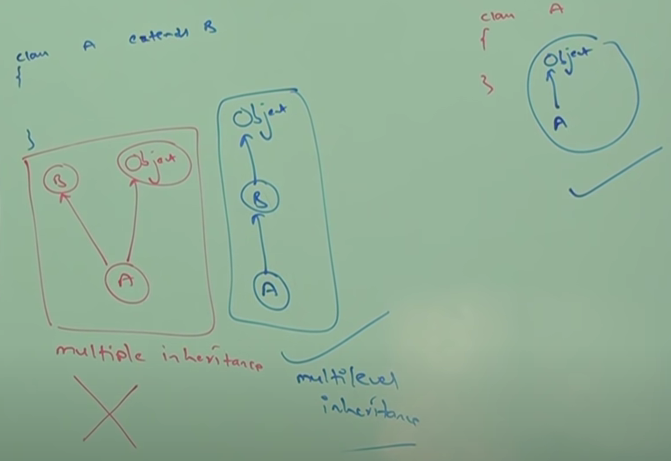
Parent reference can be used to child reference but by using that reference we can’t call child specific method but we can call parent method.

Child ref can’t be used to hold parent reference.

* The total java api is based on inheritance concept. The most common method which are applicable for any java object are defined in Object class and hence every class in the java is the child class of object either directly or indirectly so that Object class method are available to every Java class without rewriting. Due to this Object class acts as root class for every classes.

**Multiple inheritance:**

A java class can’t extends more than one class at a time hence java won’t provide support for multiple inheritance.

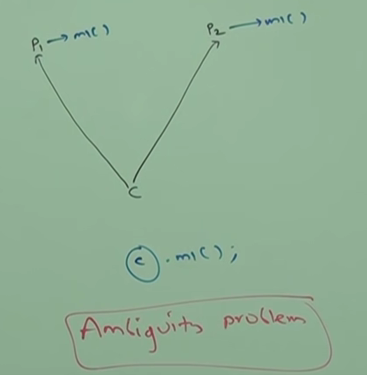


**Note:**  if our class doesn’t extends any other class then only our class is directly child class of object.

If our class extends any other class then our class indirect child class of object. This is called multilevel inheritance .

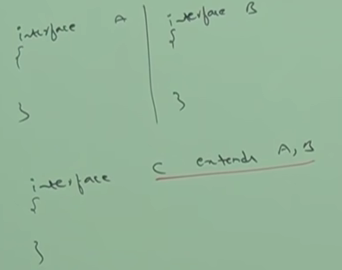
Either directly or indirectly java won’t provide support for multiple inheritance.

Q. why java won’t provide support for multiple inheritance.

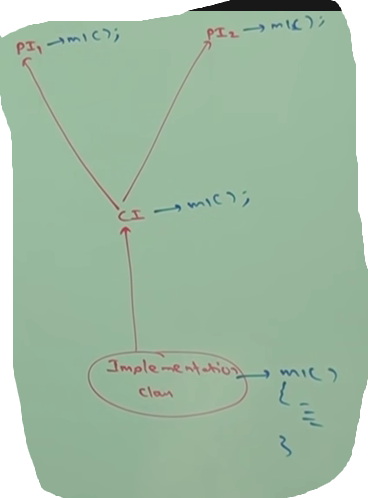


There may be a chance of ambiguity problem hence java won’t provide support for multiple inheritance.

But interface can extend any number of interface hence java provide multiple inheritance for interfaces.



Q. why ambiguity problem won’t be there in interface?

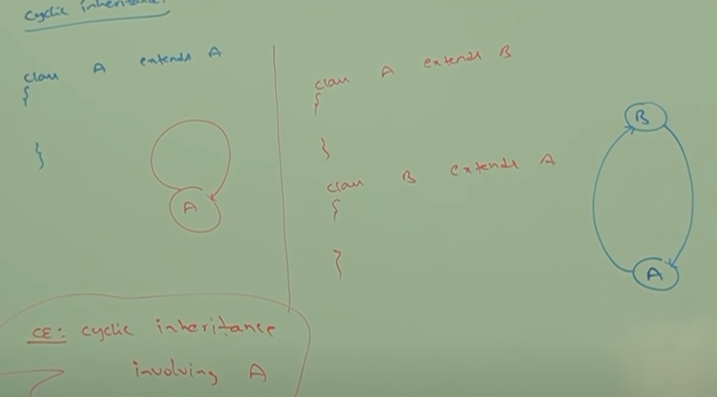


Even though multiple method declaration are available but implementation is unique and hence there is no chances of ambiguity problem in interfaces.

Note: Strictly through interfaces we won’t get any inheritance.

**Cyclic inheritance:**

It is not allowed in java. It is not required also.



**Has-A Relationship: (Composition / Aggregation )**

Has-A relationship is also known as composition or aggregation. There is no specific keyword for has a relationship but most of the time we are depending on **new** keyword.

The main advantage of has-a relationship reusability of the code.

Ex: Car has-A Engine reference

class Car{  
 Engine e = new Engine();  
}  
class Engine{  
 //engine specific functionality  
}

**Diff between composition and aggregation:**

**Composition:** Without existing container object if there no chance of existing contained object then both are associated with strong association and this strong association is composition.

Ex: University consist of several dept. without university there is no chance of dept. hence both are strongly associated hence this strong association is composition.

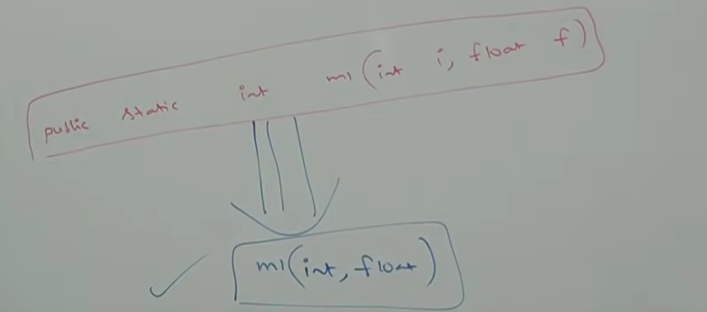
**Aggregation:** without existing container object if there is a chance of existing contained object then both are associated with weak association and this weak association is aggregation.

Ex: In a department there may be several professors, with out dept. there may be a chance of existing of professor so they are weakly associated so it is aggregation.

|  |  |
| --- | --- |
| **Composition** | **Aggregation** |
| 1. Strong association between container and contained object. 2. Container object holds directly contained object. | 1. Weak association between container and contained object. 2. Container object holds reference of contained object. |

**Method signature:**

In java method signature consists of method name followed by argument type.



Return type is not part of method signature in java.

Compiler will use method signature to resolve method calls.

**Overloading:**

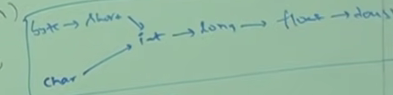
Two methods are said to be overloaded if they have same name but different argument type.

Having overloading concept in java reduces complexity of programming.

In overloading method resolution always takes care by compiler based on reference type. Hence overloading is also considered as **compile** **time** **polymorphism** or **static** **polymorphism** or **early binding**.

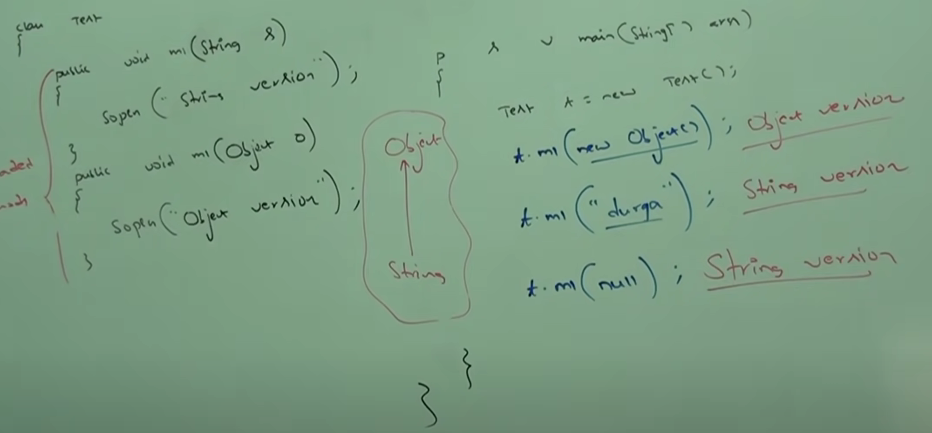
**Case-1: Automatic promotion in overloading**

While resolving overloaded method if exact match not found then immediately we won’t get any CE. First it will promote argument to the next level then it match. If matched method is available then it will be considered other compiler promotes arguments once again to the next level this process will be continued until all possible promotion still the matching not possible then we will get CE.



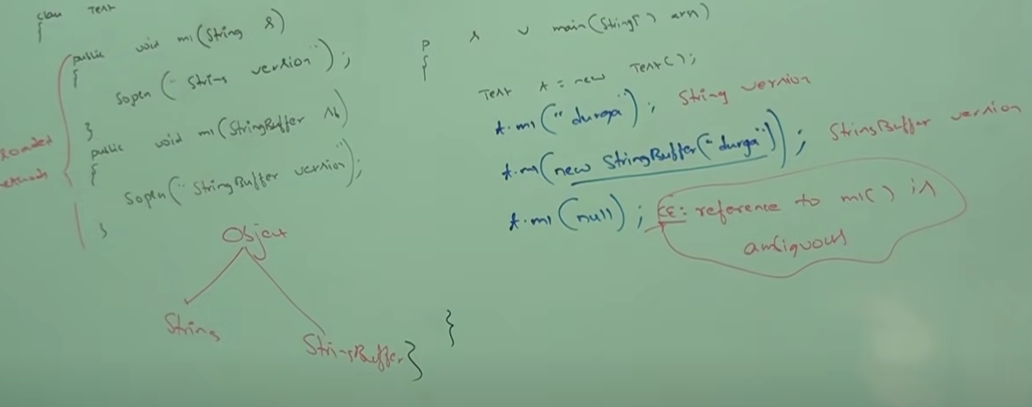
This process is called automatic promotion in overloading.

**Case- 2:**

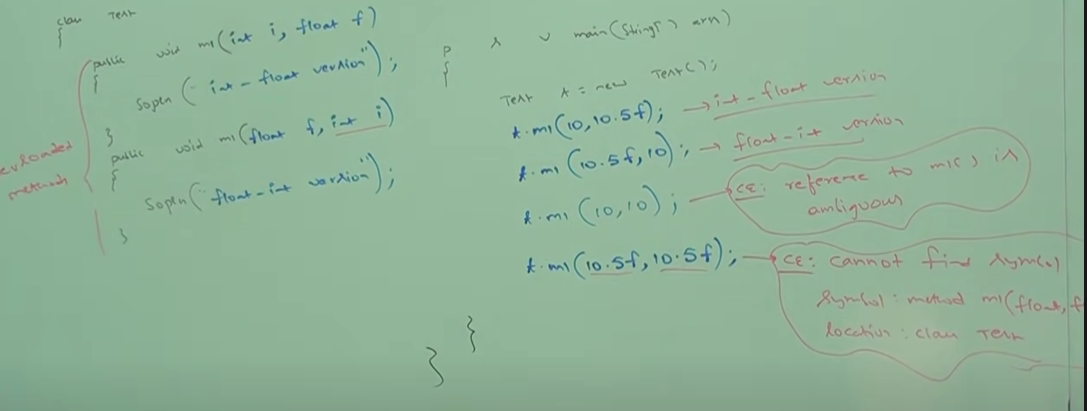
****

**Note:** while resolving overloaded method compiler will always give preference to child argument when compared with parent type argument.

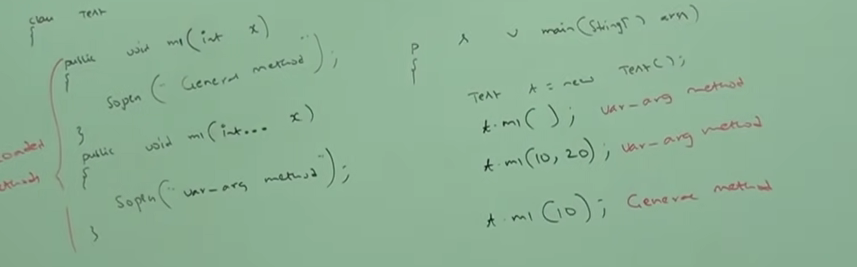
**Case- 3:**



**Case- 4:**

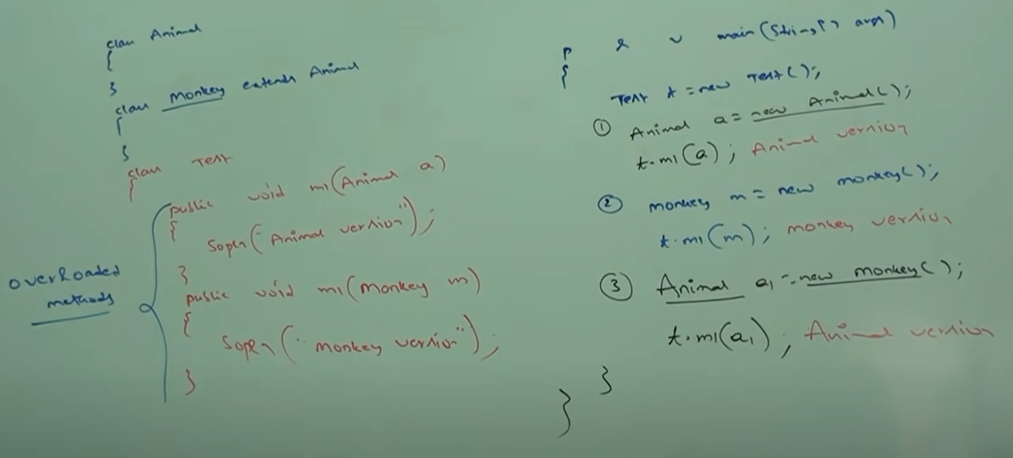


**Case- 5:**

****

In general var-arg method will get least priority it is exactly same as default case inside switch.

**Case- 6:**

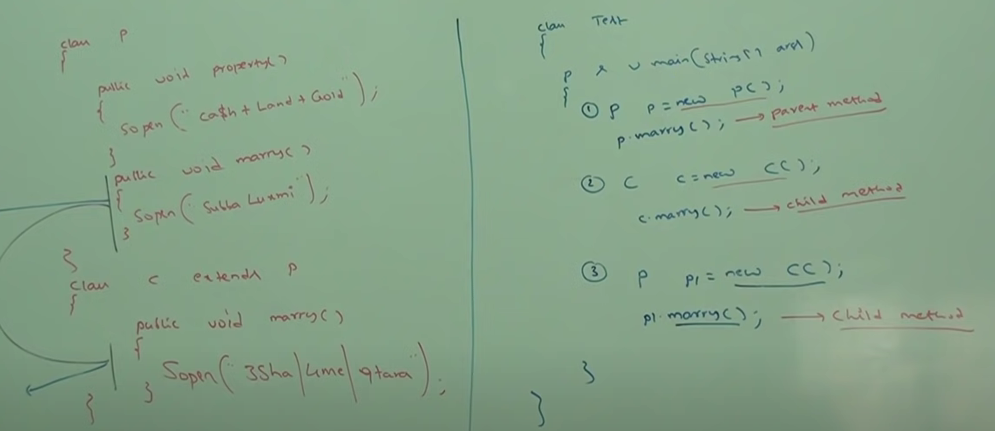
****

In overloading method resolution always takes care by compiler based on reference type. In overloading runtime object won’t play any role.

**Overriding:**

Whatever method parents have by default available to the child through inheritance, if child class not satisfied with parent class implementation then child class is allowed to redefine that method based on its requirement. This process is called overriding.

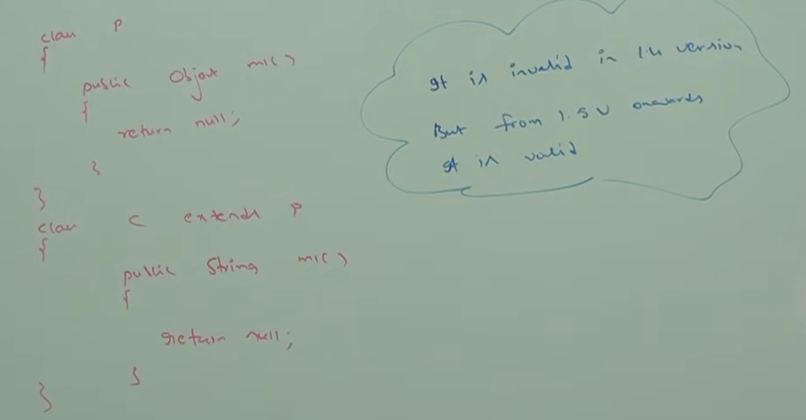
The parent class method which is overridden is called overridden method and the child class method which is overriding is called overriding method.



**Note:** In overriding method resolution always take care by JVM based on runtime object and hence overriding is also considered as runtime polymorphism, dynamic polymorphism or late binding.

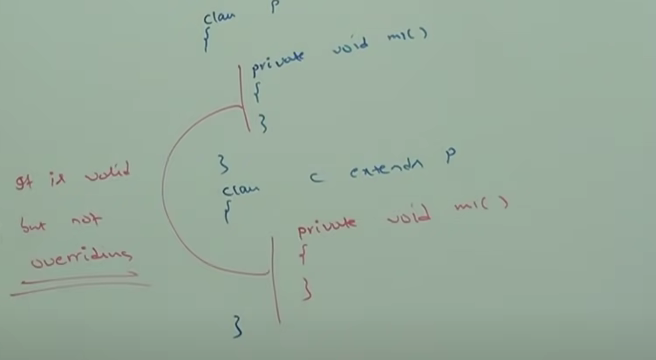
**Rules for Overriding:**

* In overriding method name and argument type must be matched i.e. method signature must be same.
* In overriding return types must be sam but this rule is applicable for 1.4 version only. For 1.5 version onwards we can take covrient return types. According to this child class return type need not be same as parent method return type its child class is allowed.

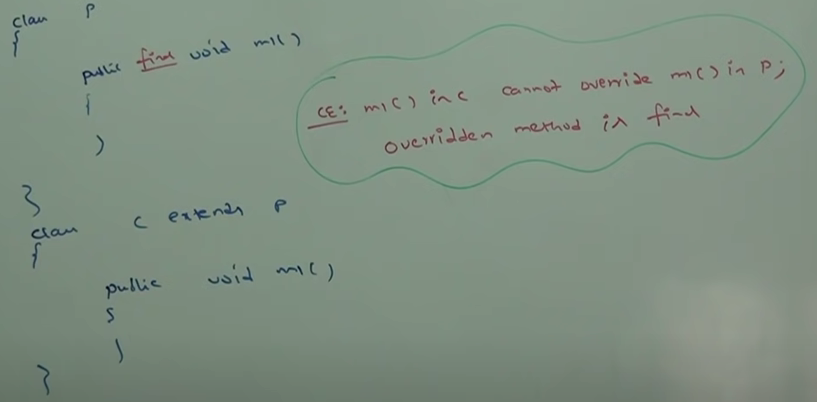


* Co-varient return type concept applicable only for object types not for primitive types.
* Parent class private method not available to the child and hence overriding concept is not applicable for private methods.

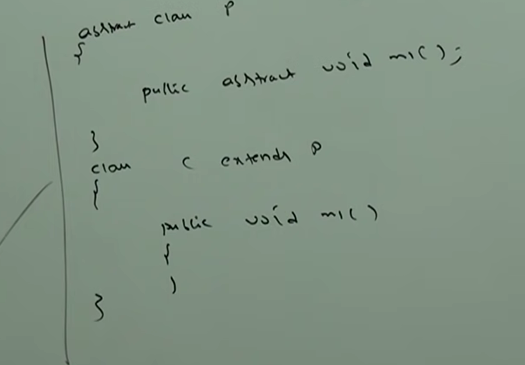
Based on our requirement it is valid but not overriding.



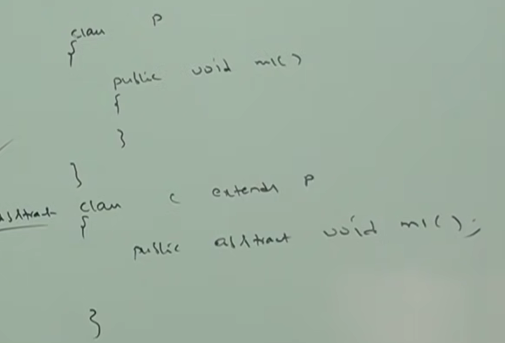
* We cant override parent class **final** methods in child classes. If we want to override then we will get compile time error.

 ­­­­­

* Parent class abstract method we should override in child class to provide implementation.



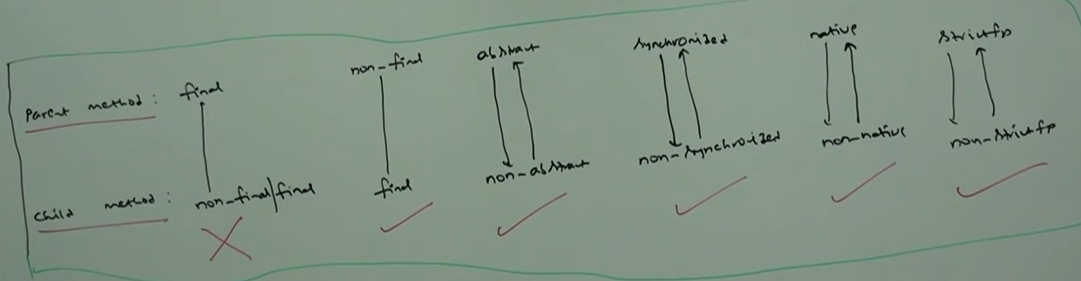
* We can override non abstract method as abstract.



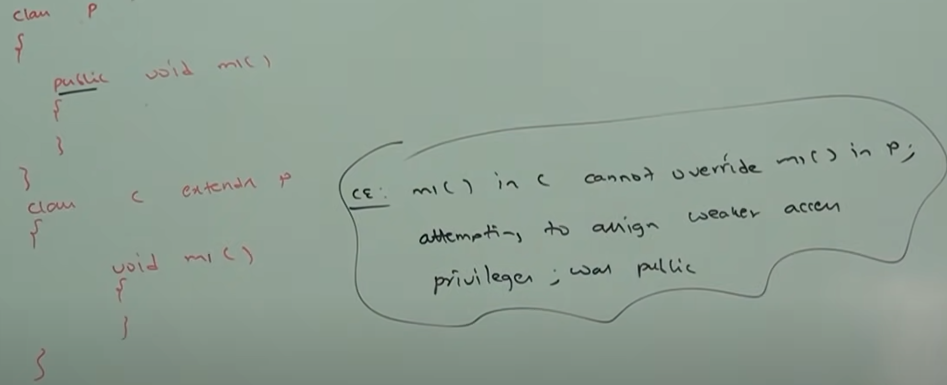
Advantage is we can stop the implementation of parent class implementation to the next level child classes.

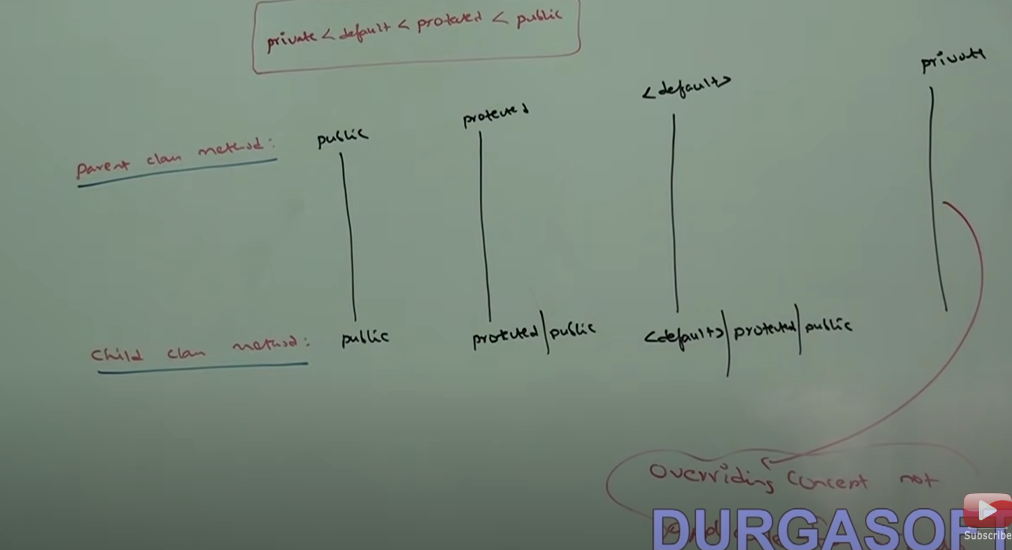
* In overriding the following modifier won’t keep any restriction.

Synchronized, native, strictfp

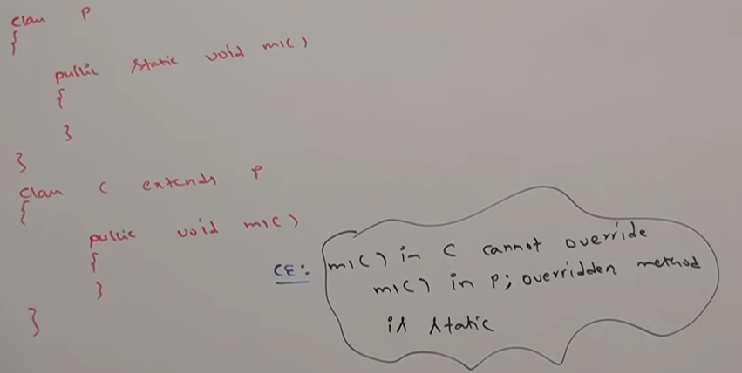


* While overriding we can’t reduce scope of access modifier but we can increase.

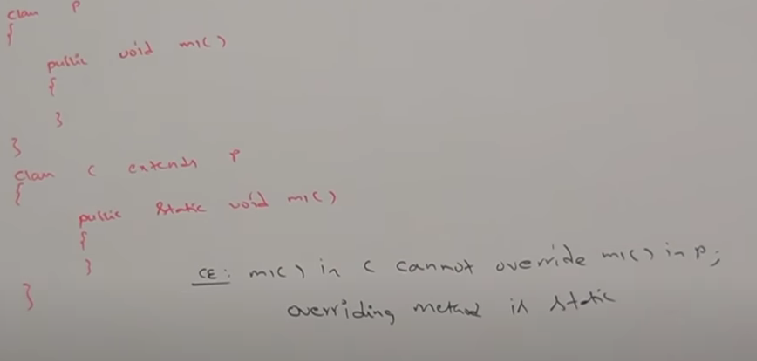




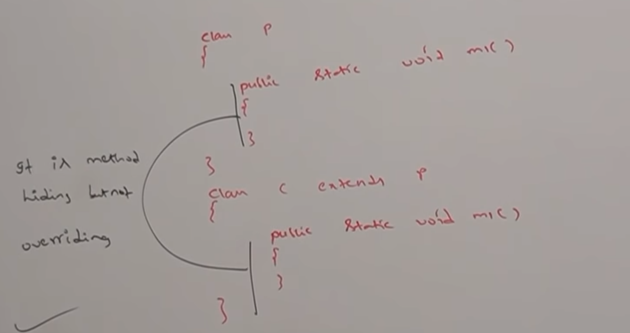
* If child class method throws any checked exception compulsory parent class method should through that same checked exception or its parent class exception otherwise we will get CE. But there is no restriction of unchecked.
* We can’t override a static method as non-static otherwise we will get CE.



* We can’t override a non-static method as static.



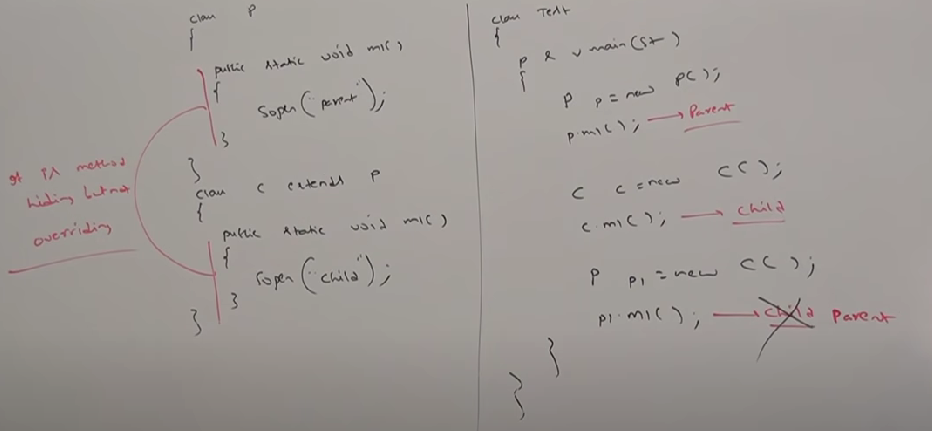
Note: if both parent and child class method are static then we won’t get any CE. It seems overriding concept is applicable for static methods but it is not overriding it is method hiding.



**Method Hiding:**

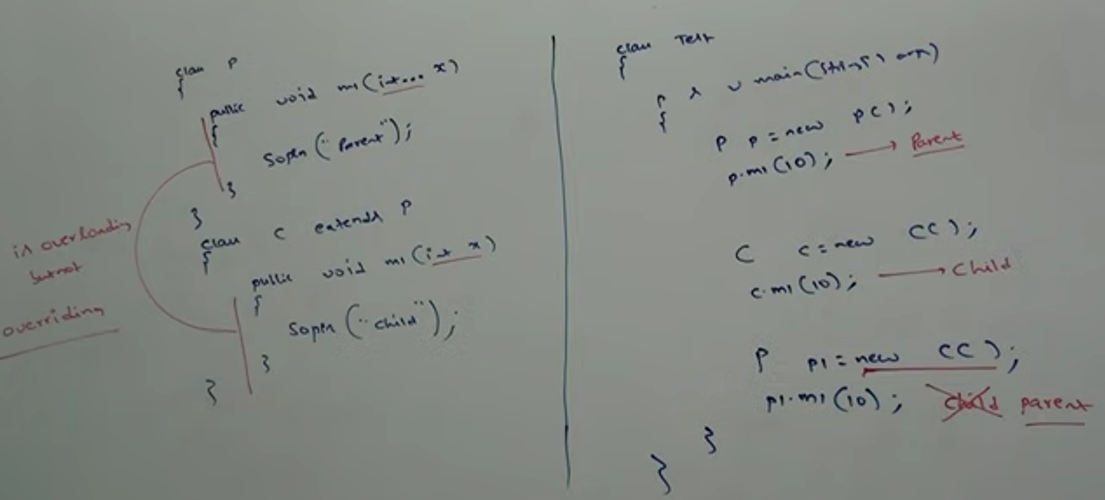
All rules of method hiding are exactly same as method overriding except the following differences:

|  |  |
| --- | --- |
| **Method hiding** | **Overriding** |
| 1. Both parent and child class method should be static. 2. Compiler is responsible for method resolution based on reference type. 3. Compile time polymorphism or static polymorphism or early binding. | 1. Both parent and child class method should be non-static. 2. JVM is always responsible for method resolution based on run time object. 3. Run time polymorphism or Dynamic polymorphism or late binding |



**Overriding in var-arg method:**

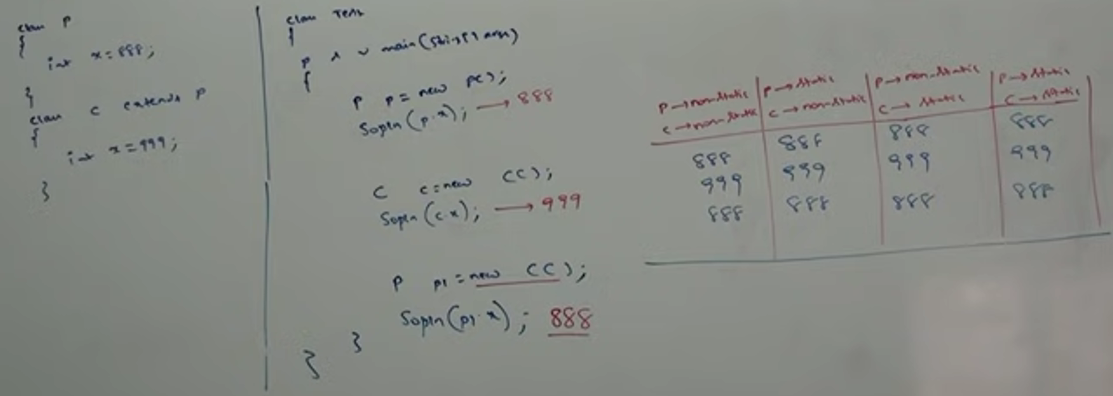
We can override var-arg method with another var-arg method only. If we are trying to override with normal method then it will become overloading not overriding.



In the above program if we replace the child method with vr-arg method then it will become overriding. In this case O/P: Parent Child Child

**Overriding with respect to variables:**

Overriding concept is not applicable for variables. Variable resolution always takes by compiler based on reference type irrespective of whether the variable is static or non-static.



**Difference between Overloading and Overriding:**

|  |  |  |
| --- | --- | --- |
| **Property** | **Overloading** | **Overriding** |
| 1. Method names 2. Argument types 3. Method signatures 4. Return types 5. Private static and final methods 6. Access Modifier 7. Throws class 8. Method resolution 9. It is also known as | 1. Must be same. 2. Must be different(At least order). 3. Must be different. 4. No restrictions. 5. Can be overloaded. 6. No restrictions. 7. No restrictions. 8. Takes care by compiler based on reference type. 9. Compile time polymorphism / static polymorphism / early binding. | 1. Must be same. 2. Must be same (including order). 3. Must be same. 4. Must be same until 1.4 version but from 1.5v onwards co variant return types are allowed. 5. Can’t be overridden. 6. We can’t scope of access modifier but we can increase. 7. If child class method thrown checked exception then compulsory parent class method should throw the same checked exception or its parent, but no restrictions for unchecked exception. 8. Always takes care by JVM based on runtime object. 9. Runtime polymorphism / dynamic polymorphism / late binding. |

**Note:** in overloading we haver to check only method names (must be same) and argument type(must be different). We are not required to check remaining like return types access modifiers etc.

But in overriding every thing we have to check method name argument types, return type, throws class etc.